PERSEPHONE: Zero-Knowledge Multiplicative Non-Negative Proof for Sequential Private Range Verification

Ivan Tjuawinata[1], Yann Fraboni[2], Jun Jie Sim[2], Darian Gunamardi[1], Zhenghao Wu[2], Hasventhran Baskaran[1], Chi-Hung Chi[1], Yujing Sun[1†], Pu Duan[2], Kwok-Yan Lam[1]
[1] Nanyang Technological University
[2] Ant International, Ant Group

Abstract

Privacy is a fundamental requirement of modern digital payment systems, which process highly sensitive financial data, including account balances and transaction amounts. Unauthorized disclosure of this information can compromise users' financial privacy and expose them to fraud or other forms of financial abuse. A key challenge in this setting is to verify that a payment amount does not exceed the payer's available balance without revealing either the payment amount or the balance. A common approach is to employ zero-knowledge range proofs (ZKRPs). However, because conventional ZKRPs require public bounds, verifying a private value against private bounds requires multiple ZKRP instances, resulting in substantial computational and communication overhead that can increase latency and lead to cascading failures at scale. Motivated by the sequential nature of transactions in real-world payment systems, we formalize the problem of *Sequential Private Range Verification* (SPRV) and propose a framework, PERSEPHONE, that exploits the sequential structure of linked verifications rather than treating each verification independently. Within PERSEPHONE, we design and leverage a novel zero-knowledge proof primitive, ZK-MULTNNP, to solve SPRV efficiently. Experimental results demonstrate that PERSEPHONE reduces total proving time by at least 3× compared to a ZKRP-based baseline. Furthermore, in a digital payment scenario, PERSEPHONE completes all transaction verifications within the 400 ms Doherty threshold, compared to only 30% of verifications for the ZKRP-based baseline.

Index Terms

Zero-knowledge proof, Pedersen commitment scheme, non-negativity proof

I. INTRODUCTION

Modern digital payment systems handle highly sensitive financial information, such as account balances and transaction amounts. Unauthorized disclosure of such information compromises users' financial privacy and may facilitate fraud or other forms of financial abuse. Therefore, it is essential for payment services to operate while preserving the confidentiality of customers' sensitive financial information. Concretely, to validate a payment request, a payment system should be able to verify that a customer's account balance is sufficient to cover a transfer amount without revealing either the account balance or the transfer amount. Such privacy-preserving functionality is essential for protecting users' financial information while maintaining the correctness and security of payment transactions.

Zero-knowledge proofs (ZKPs) offer a practical mechanism for implementing privacy-preserving payment systems because they enable transaction correctness to be verified without exposing users' sensitive financial information [1]. Figure 1 illustrates a typical use case in a digital payment system. A user initiates a payment request and then engages in a zero-knowledge verification protocol with the payment system to prove that the transferred amount is valid. Upon successful verification, the system approves the transaction.

Nevertheless, simultaneously ensuring transaction validity and preserving confidentiality is nontrivial. When verifying a payment amount, the payer's account balance, which serves as the verification upper bound, must remain confidential. However, conventional zero-knowledge range proofs (ZKRPs) [2, 3] are designed to verify that a secret value lies within specified *public* bounds. Therefore, they cannot be directly applied when the verification bounds themselves are private.

In real-world digital payment systems, transactions associated with a given account are typically processed sequentially because each transaction updates the account state and subsequent transactions must operate on the resulting balance. We refer to the problem of verifying a sequence of values against sequentially updated private bounds as ***Sequential Private Range Verification*** (SPRV). We refer to each individual verification in the sequence as a *private range verification* (PRV).

A straightforward approach is to verify each transaction independently. Since conventional ZKRPs require public bounds, each private-range verification must be decomposed into multiple ZKRP instances with public bounds. In our construction, we show that four ZKRP calls are required for each private range verification under this independent verification approach. This decomposition introduces substantial computational and communication overhead by increasing the number of proof operations required per transaction. Moreover, the public bounds must be sufficiently large to encompass any possible account balance, even though such extreme values are unlikely to occur in practice. Consequently, independently verifying each transaction incurs considerable cost, making this approach particularly impractical for long sequences of transactions.

† denote corresponding author with email yujing.sun@ntu.edu.sg

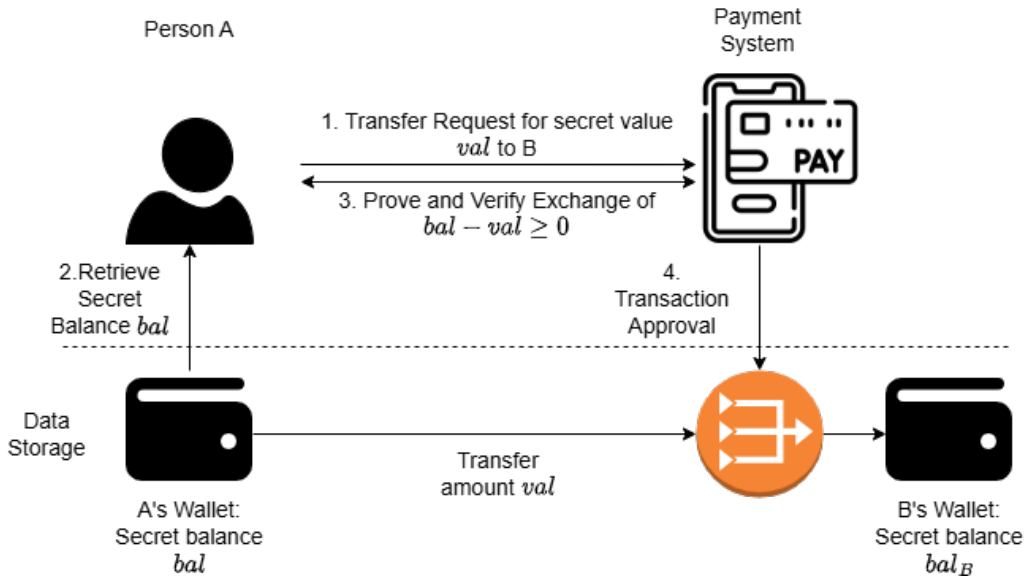

Fig. 1. Illustration of a real-world privacy-preserving digital payment system.

In this work, we introduce PERSEPHONE, a novel solution to the SPRV problem with applications to digital payment systems and risk assessment platforms. As its first step, PERSEPHONE reformulates each private range verification in SPRV as three multiplicative non-negativity relations, reducing the number of relations from four to three. A multiplicative non-negativity relation asserts that the product of two secret values is non-negative. PERSEPHONE further exploits the inherent sequential structure of digital payment transactions to significantly reduce the computational and communication overhead of private range verification. In real-world digital payment systems, account balances are inherently dependent across consecutive transactions: the balance after one transaction is determined by the previous balance and the corresponding transfer amount. Specifically, SPRV entails verifying a sequence of private range constraints in which the bounds for each subsequent verification are updated as linear combinations of the previous bounds and the corresponding input value. Motivated by this observation, rather than treating each transaction as an independent range verification, PERSEPHONE takes advantage of the sequential dependency between transactions. By linking consecutive PRVs through the previously verified bounds, the two multiplicative non-negativity relations concerning the validity of these bounds no longer need to be proven at each subsequent step, reducing the number of multiplicative non-negativity proofs to one.

A. Our Contributions

Our contributions are as follows.

- We reduce each private range verification in SPRV from four standard range proofs to three multiplicative non-negativity proofs. Specifically, proving that $0 \leq \mathtt{l} \leq x \leq \mathtt{u} \leq B$ for secret values $\mathtt{l}, x, \mathtt{u}$ and public bound B is equivalent to proving three non-negativity conditions on products of two values, with two depending on B and one being independent of B. This reformulation replaces four B-dependent ZKRPs with two B-dependent ZK-MULTNNPs of comparable cost and one less expensive B-independent ZK-MULTNNP. To this end, we introduce a *multiplicative non-negativity* relation, which captures statements of the form $w_1 w_2 \geq 0$ given commitments of w_1 and w_2. To the best of our knowledge, the multiplicative non-negativity relation has not previously been considered as a dedicated relation for zero-knowledge proofs.
- We realize a zero-knowledge proof for the multiplicative non-negativity relation, ZK-MULTNNP, by adapting the CKLR-style zero-knowledge range-proof construction [4], which uses a three-square decomposition.
- We propose a stateful protocol based on a chain-of-verified-commitments that exploits the sequential structure of the SPRV problem to eliminate the two B-dependent proofs entirely, reducing each verification step to a single B-independent ZK-MULTNNP.

In practice, PERSEPHONE solves the SPRV problem 3× faster than the ZKRP-based solution. Specifically, in our proof-of-concept transaction system, all verifications using PERSEPHONE complete within the 400 ms Doherty threshold [5], whereas only 30% of verifications using the ZKRP-based solution do so.

B. Related Work

Confidential transactions, first proposed by Maxwell [3], use cryptographic commitments in place of transaction amounts together with zero-knowledge range proofs (ZKRPs) to ensure privacy. Earlier work on privacy-preserving payments includes

ZeroCash [6], which used ZK-SNARKs to hide both transaction amounts and balances. Several systems implement confidential transactions in practice, including Aztec [7], Zether [2], and [8], all of which rely on ZKRPs to verify transaction validity. We refer to [9] for a comprehensive survey and briefly review the main classes of ZKRPs below. DCAP considered a complementary decentralized-payment setting, using signatures of knowledge to provide conditional transaction anonymity while retaining the ability to trace suspicious transactions [10].

Bulletproofs and its variants [11–13] are widely adopted for confidential transactions because they offer logarithmic proof sizes and require no trusted setup. They use the binary representation of a b-bit secret value to prove that it lies within the range $[0, 2^b - 1]$. Verange [14] provides additional binary-decomposition variants optimized for verifier complexity, prover complexity, and proof size, respectively. Since inner-product arguments are central components of many compact range-proof constructions, improvements to their verification complexity are also relevant. Leopard provided logarithmic proof size and sublinear verification under the discrete-logarithm assumption without requiring a trusted setup [15].

A different class of ZKRPs relies on three-square decomposition [4]. This approach reformulates a range proof as a non-negativity proof by showing that a quantity derived from the secret value can be expressed as a sum of three squares. Sharp [16] further improves this approach by introducing batching techniques and adapting the construction to the Σ-protocol framework.

A third class relies on hash chains [17], where the proof that a committed value exceeds a threshold is constructed by iterated hashing. While conceptually simple, this approach requires exponential time for general ranges. [17] reduces this to linear complexity through a decomposition technique. More recently, Li et al. proposed SHARP-PQ, a succinct hash-based arbitrary-range proof that provides post-quantum security, polylogarithmic proof size, and optimized batch proofs [18].

Beyond specialized range-proof protocols, general-purpose transparent zero-knowledge frameworks have also been developed. Zilch facilitates the deployment of zero-knowledge verifiable computation using an instruction-based programming model and does not require a trusted setup [19]. Such frameworks offer greater generality, whereas Persephone develops a specialized proof relation and stateful protocol for efficient sequential private-range verification.

II. Preliminaries

In Section II-A, we recall the Pedersen commitment scheme used throughout this work. In Section II-B, we define the range and non-negative relations.

A. Commitment Scheme

Definition 1 (Commitment with refresh mechanism): A scheme $\texttt{Com}$ with a refresh mechanism consists of a 3-tuple of algorithms $(\texttt{Setup}, \texttt{Commit}, \texttt{Refresh})$ such that

- $\mathsf{pp} \leftarrow \texttt{Com.Setup}(1^\lambda)$: Given a security parameter λ, returns public parameters pp.
- $\mathsf{c}_x \leftarrow \texttt{Com.Commit}(\mathsf{pp}, x, r)$: Given public parameters pp, message x and randomness r, returns commitment c_x. For simplicity of notation, we will omit r whenever it is not essential and its meaning is clear from the context.
- $\{0, 1\} \leftarrow \texttt{Com.Refresh}((\mathsf{pp}, \mathsf{c}', \mathsf{c}); (r', r))$: Given public parameters pp, and two commitments c' and c with respective randomness r' and r, returns 1 if c' is a valid commitment of the same value as c and 0 otherwise.

In general, the refresh algorithm is not part of the standard definition of a commitment scheme. In this work, we add this requirement for the prover to refresh the randomness of its commitments and allow valid recommitments of verified values.

For a commitment c_x of a message x, we write $\overline{\mathsf{c}_x}$ to denote a commitment that has been verified to satisfy a relation, which consists of prepending an additional bit in practice.

In this work, we instantiate the commitment scheme with the Pedersen commitment, which we briefly recall: $\texttt{Com.Setup}$ returns a cyclic group $\mathbb{G}$ of prime order p and two generators g and h; $\texttt{Com.Commit}$ returns a commitment $\mathsf{c} = g^x h^r$ for some random $r \in \mathbb{Z}_p$. For a public value $A \in \mathbb{Z}_p$, we construct its commitment by setting the randomness to 0, i.e. $r = 0$ and $\mathsf{c}_A = g^A$. The Pedersen commitment scheme is additively homomorphic by construction of $\texttt{Com.Commit}$. Given two values x_1 and x_2 with respective commitments c_1 and c_2, a valid commitment to any linear combination $\lambda x_1 + \mu x_2$, for arbitrary $\lambda, \mu \in \mathbb{Z}_p$, can be constructed as $\mathsf{c}_1^\lambda \mathsf{c}_2^\mu = g^{\lambda x_1 + \mu x_2} h^{\lambda r_1 + \mu r_2}$. For completeness, we provide a way to realize the commitment refresh functionality in the Supplemental Material A, which follows [20, Construction 2.2]. We note that despite our focus on Pedersen commitment, Persephone can also be realized using other additive homomorphic commitment schemes such as exponential ElGamal [21] and Fujisaki-Okamoto [22].

Since the Pedersen commitment scheme operates over $\mathbb{Z}_p$, we require the following assumption to ensure that arithmetic over integers is faithfully captured.

Assumption 1 (No Wrap-Around): All values in this work are integers embedded in $\mathbb{Z}_p$, where p is the order of the underlying group $\mathbb{G}$. We assume p is sufficiently large that all intermediate values arising during protocol execution remain within $[-M, M]$ for some $M \ll p$, so that no modular wrap-around occurs. Under this assumption, the embedding from integers to $\mathbb{Z}_p$ is unambiguous, and non-negativity conditions over the integers are preserved under the group representation.

B. Zero Knowledge Proof

Let $\mathcal{R}$ be a relation containing tuples $(z; w)$, such that, given z and w, we can verify whether $(z; w) \in \mathcal{R}$ in polynomial time. A zero-knowledge proof (ZKP) for $\mathcal{R}$ is a protocol where, given z, a prover $\mathcal{P}$ convinces a verifier $\mathcal{V}$ that $\mathcal{P}$ knows a *witness* w such that $(z; w) \in \mathcal{R}$ and such that the ZKP protocol satisfies completeness, soundness, and zero-knowledge. We defer the reader to [9] for a more detailed description of ZKPs.

We define the range relation $\mathcal{R}_{\mathtt{Range}}$. Given a commitment scheme $\mathtt{Com}$ for values in $\mathbb{Z}$, a pair $a, b \in \mathbb{Z}$ such that $a \leq b$, and public parameters pp, the range relation with respect to (a, b), is defined as

$$\mathcal{R}_{\mathtt{Range}}(a, b) = \{ (\mathsf{c}; x) : \mathsf{c} = \mathtt{Com.Commit}(\mathsf{pp}, \mathsf{x}) \text{ and } a \leq x \leq b \}. \tag{1}$$

Throughout this work, we refer to $x \in [a, b]$ instead of $a \leq x \leq b$, where $[a, b]$ denotes the set of integers from a to b, i.e. $[a, b] = \{a, a + 1, \ldots, b\}$.

We briefly describe the zero-knowledge range proof (ZKRP) proposed in [4, Section 5.3] for this relation. By noting that $x \in [a, b]$ is equivalent to $\tilde{x} \in [0, \tilde{b}]$, where $\tilde{x} = x - a$ and $\tilde{b} = b - a$, we can focus on the case where the input range is $[0, \tilde{b}]$. The proof uses Legendre's three-square theorem to express $4\tilde{x}(\tilde{b} - \tilde{x}) + 1$ as a sum of three squares of integers. If these squares of integers exist, then $\tilde{x}(\tilde{b} - \tilde{x}) \geq 0$. Since $\tilde{b}$ is public and known to be non-negative, we have $\tilde{x} \geq 0$ and $\tilde{b} - \tilde{x} \geq 0$, and equivalently $a \leq x \leq b$. We refer the reader to the Supplemental Material E for a more detailed description. In this work, we denote the functionality responsible for determining three non-negative integers y_1, y_2, and y_3 such that $w = y_1^2 + y_2^2 + y_3^2$ by $(y_1, y_2, y_3) \leftarrow \mathcal{F}^{(3\mathsf{SP})}(w)$.

A relation that is similar to the range relation is the non-negativity relation $\mathcal{R}_{\mathtt{NonNeg}}$.

$$\mathcal{R}_{\mathtt{NonNeg}} = \{ (\mathsf{c}; x) : \mathsf{c} = \mathtt{Com.Commit}(\mathsf{pp}, \mathsf{x}) \text{ and } x \geq 0 \}. \tag{2}$$

Given a universal upper bound $\mathcal{U}$ such that $x \leq \mathcal{U}$ for all possible values x, a ZKP for $\mathcal{R}_{\mathtt{NonNeg}}$ can be realised as a ZKRP with $(a, b) = (0, \mathcal{U})$.

III. Sequential Private Range Verification

We consider a setting in which the verifier performs a sequence of T verifications. The verifications are processed sequentially, meaning that proof generation and state updates for each step are completed before the next step begins. At every step t, the prover holds a value x_t and bounds (l_{t-1}, u_{t-1}) carried over from the previous step and proves to the verifier that x_t lies within these bounds. Since all values and bounds must also lie within $[0, B]$, this amounts to proving the ordered chain of inequalities

$$0 \leq l_{t-1} \leq x_t \leq u_{t-1} \leq B. \tag{3}$$

The bounds for the next step are determined as follows. If Equation (3) is not satisfied, then the bounds remain unchanged, i.e., $(l_t, u_t) = (l_{t-1}, u_{t-1})$. Otherwise, the bounds are updated using a publicly known function f_{update} whose two output components are linear combinations of the prover's current value and bounds.

$$(l_t, u_t) = f_{update}(l_{t-1}, x_t, u_{t-1}). \tag{4}$$

We also assume that f_{update} is defined so that whenever Equation (3) holds, the updated bounds satisfy

$$0 \leq l_t \leq u_t \leq B. \tag{5}$$

For simplicity, we use the same update function f_{update} at every step. Allowing a different publicly known update function at each step would not affect either the baseline or our solution.

The problem statement can be generalized to include steps at which the prover provides no input, denoted by $x_t = \perp$. In this case, the bounds are carried forward unchanged, with $l_t = l_{t-1}$ and $u_t = u_{t-1}$. For the remainder of this work, we assume for simplicity that the prover provides a value at every step.

A. Security Definition

A protocol that realizes this functionality must satisfy the following properties:

- **Completeness**: If Equation (3) is satisfied at every iteration $t = 1, \ldots, T$, and bounds are updated according to f_{update}, then an honest execution is accepted with probability 1.
- **Soundness**: If $0 \leq l_0 \leq u_0 \leq B$ does not hold, the verifier rejects except with negligible probability. Furthermore, at every iteration $t = 1, \ldots, T$, if either Equation (3) does not hold or the updated bounds do not follow the update rule in Equation (4), the verifier rejects at iteration t except with negligible probability.
- **Honest-verifier zero-knowledge (HVZK)**: For any honest PPT verifier $\mathcal{V}$, there exists a PPT simulator $\mathcal{S}$ that, given only the public parameters and the sequence of accept or reject outcomes, generates a transcript computationally indistinguishable from a real execution. The simulated transcript reveals nothing about the private bounds $\{(l_t, u_t)\}_{t=0}^{T}$ or private values $\{x_t\}_{t=1}^{T}$ beyond the public outcomes.

B. A Naive Solution

Equation (3) can be proven using multiple ZKRP instances, as in the simplified setting without a lower bound considered in [2, 7, 8]. Each step is proven independently, so this approach does not exploit the relationship between bounds across steps defined by the update function f_{update} in Equation (4). Hence, the sequence of T steps can be considered as T successive independent verifications of Equation (3). At each step, the verifier verifies the following relation defined for four positive integers $\mathtt{l}$, x, $\mathtt{u}$, and B as

$$\begin{aligned}\mathcal{R}_{\text{PRV}}(B) = \{ ((\mathtt{c_l}, \mathtt{c}_x, \mathtt{c_u}); (\mathtt{l}, x, \mathtt{u}, r_\mathtt{l}, r_x, r_\mathtt{u})) : \\ \mathtt{c_l} = \mathtt{Com.Commit}(\mathtt{pp}, \mathtt{l}, r_\mathtt{l}), \\ \mathtt{c}_x = \mathtt{Com.Commit}(\mathtt{pp}, x, r_x), \\ \mathtt{c_u} = \mathtt{Com.Commit}(\mathtt{pp}, \mathtt{u}, r_\mathtt{u}), \\ \text{and } 0 \le \mathtt{l} \le x \le \mathtt{u} \le B \}. \end{aligned} \tag{6}$$

Relation (6) extends the range relation in Equation (1) from a single committed value to an ordered sequence of three committed values between 0 and B. It can also be viewed as a private range verification in which the commitments to the lower and upper bounds have not been verified previously. We say that the triple of commitments (c_l, c_x, c_u) is a valid triple if it consists of commitments to l, x, u satisfying $0 \le l \le x \le u \le B$. In this case, $((\mathtt{c_l}, \mathtt{c}_x, \mathtt{c_u}); (\mathtt{l}, x, \mathtt{u}, r_\mathtt{l}, r_x, r_\mathtt{u})) \in \mathcal{R}_{\text{PRV}}(B)$.

At every step, the prover convinces the verifier that its triple of commitments $(c_{l_{t-1}}, c_{x_t}, c_{u_{t-1}})$ is valid according to Relation (6). A straightforward implementation invokes a ZKRP once for each of the following four ranges:

$$\begin{aligned} &0 \le l_{t-1} \le B, && 0 \le x_t - l_{t-1} \le B, \\ &0 \le u_{t-1} - x_t \le B, && \text{and } 0 \le u_{t-1} \le B, \end{aligned} \tag{7}$$

where $l_{t-1} \le u_{t-1}$ is implied by the second and third inequalities. The prover can then update the bounds locally for the next iteration according to Equation (4). We refer to this solution throughout this work as the baseline. The main limitation of this approach is that every PRV requires four range proofs using the global bound B, which can be much larger than the private upper bound u_{t-1} used at step t, making these proofs unnecessarily slow.

IV. Reduction of SPRV to Multiplicative Non-Negativity Relations

In Section IV-A, we establish intermediate results that are used in Section IV-B to derive an equivalent formulation of Equation (3). In Section IV-C, we show how this reformulation enables faster verification.

A. Intermediate Results

We introduce Lemma 1, which shows that a range inequality can be equivalent to a non-negative condition.

Lemma 1: For real numbers α, β, and v satisfying $\alpha \le \beta$, we have $\alpha \le v \le \beta$ if and only if $(v - \alpha)(\beta - v) \ge 0$.

Proof. If $v \in [\alpha, \beta]$, then both $v - \alpha$ and $\beta - v$ are nonnegative, so their product is nonnegative. If $v < \alpha$, then $v - \alpha < 0$ and $\beta - v > 0$. If $v > \beta$, then $v - \alpha > 0$ and $\beta - v < 0$. Thus, outside $[\alpha, \beta]$, the two factors have opposite signs and their product is negative. Therefore, $(v - \alpha)(\beta - v) \ge 0$ holds if and only if $\alpha \le v \le \beta$. □

Lemma 1 characterizes whether a single value satisfies $\alpha \le v \le \beta$ using a nonnegative product condition. We now extend this characterization to sequences satisfying $\alpha \le v_1 \le \cdots \le v_n \le \beta$. By applying the lemma successively, we obtain one product condition whenever a value is placed between two values whose relative order has already been established. The following corollary shows that the values $v_1, \ldots, v_n$ may be considered in any order.

Corollary 1: Let α, β, and $v_1, \ldots, v_n$ be real numbers satisfying $\alpha \le \beta$. Set $v_0 = \alpha$ and $v_{n+1} = \beta$. Let π be a permutation of $\{1, \ldots, n\}$ that specifies the order in which $v_1, \ldots, v_n$ are considered. Set $S_0 = \{0, n+1\}$. For each $i \in \{1, \ldots, n\}$, define

$$L_i = \max\{k \in S_{i-1} : k < \pi(i)\}, \tag{8}$$

$$R_i = \min\{k \in S_{i-1} : k > \pi(i)\}, \tag{9}$$

$$S_i = S_{i-1} \cup \{\pi(i)\}. \tag{10}$$

Then $\alpha \le v_1 \le \cdots \le v_n \le \beta$ if and only if

$$\forall i \in \{1, \ldots, n\}, \qquad (v_{\pi(i)} - v_{L_i})(v_{R_i} - v_{\pi(i)}) \ge 0. \tag{11}$$

Proof. For the forward direction, assume that $\alpha \le v_1 \le \cdots \le v_n \le \beta$. For every $i \in \{1, \ldots, n\}$, the inequalities $L_i < \pi(i) < R_i$ imply $v_{L_i} \le v_{\pi(i)} \le v_{R_i}$, so both factors in the product condition are nonnegative. For the reverse direction, assume that the product condition holds for every $i \in \{1, \ldots, n\}$. We prove by induction that the values whose indices belong to S_i are nondecreasing in their index order. The base case holds for $S_0 = \{0, n+1\}$ because $v_0 = \alpha \le \beta = v_{n+1}$. Suppose that the values indexed by S_{i-1} are ordered. The indices L_i and R_i are the nearest elements of S_{i-1} on either side of $\pi(i)$, so $v_{L_i} \le v_{R_i}$. Applying Lemma 1 to v_{L_i}, v_{R_i}, and $v_{\pi(i)}$ shows that the product condition is equivalent to $v_{L_i} \le v_{\pi(i)} \le v_{R_i}$. Adding $\pi(i)$ to S_{i-1} therefore preserves the ordering. After n steps, $S_n = \{0, \ldots, n+1\}$, which gives $\alpha \le v_1 \le \cdots \le v_n \le \beta$. □

B. Novel Equivalence for Equation (3)

Equation (3), introduced in Section III, is equivalent to the four range constraints used by the baseline in Equation (7). Each range has two inequalities, and we realise that each range has a redundant inequality, which makes Equation (3) equivalent to the following four non-negativity conditions,

$$0 \le l_{t-1},\ 0 \le x_t - l_{t-1},\ 0 \le u_{t-1} - x_t,\ \text{and } 0 \le B - u_{t-1}. \tag{12}$$

To obtain a single product condition involving x_t, we apply Corollary 1 with $n = 3$, $\alpha = 0$, $\beta = B$, $(v_1, v_2, v_3) = (l_{t-1}, x_t, u_{t-1})$, and $(\pi(1), \pi(2), \pi(3)) = (1, 3, 2)$, which places x_t after the two bounds. The corollary then shows that Equation (3) is equivalent to the following three non-negativity conditions:

$$l_{t-1}(B - l_{t-1}) \ge 0,\ (u_{t-1} - l_{t-1})(B - u_{t-1}) \ge 0, \tag{13}$$

and

$$(u_{t-1} - x_t)(x_t - l_{t-1}) \ge 0. \tag{14}$$

This reformulation replaces the four range proofs required by the baseline with three instances of the common relation introduced in the next subsection, enabling a more efficient proof construction.

C. Multiplicative Non-Negativity Relation

We consider the problem of proving that $w_1 w_2 \ge 0$ for two committed values w_1 and w_2. This problem captures the three non-negativity conditions in Equations (13) and (14), where w_1 and w_2 are linear combinations of l_{t-1}, x_t, u_{t-1}, and B.

The verifier has the claimed commitments c_{w_1} and c_{w_2}, and the prover has the two values w_1 and w_2 and their randomness r_{w_1} and r_{w_2}. Proving $w_1 w_2 \ge 0$ can be done with two public range proofs as in the baseline, i.e. $w_1, w_2 \in [0, B]$. However, this requires six ZKRP calls per step instead of the four required by the baseline. If the verifier had a commitment of $w_1 w_2$, verifying $w_1 w_2 \ge 0$ would require a single ZKRP call. However, the Pedersen commitment scheme is not multiplicative, which prevents the verifier from computing this commitment with that of w_1 and w_2.

To address this issue, we introduce a relation that captures statements of the form $w_1 w_2 \ge 0$ given commitments c_{w_1} and c_{w_2}. We call it the *multiplicative non-negativity relation*. That is,

$$\begin{aligned}\mathcal{R}_{\texttt{MultNNP}} = \{ ((\mathsf{c}_{w_1}, \mathsf{c}_{w_2}); (w_1, w_2, r_1, r_2)) :\\ \mathsf{c}_{w_i} = \texttt{Com.Commit}(\mathsf{pp}, w_i, r_i) \text{ for } i \in \{1, 2\}\\ \text{and } w_1 w_2 \ge 0 \}.\end{aligned} \tag{15}$$

We refer to a zero-knowledge proof for this relation as ZK-MultNNP and present a realization obtained by adapting the construction of [4] in Section V.

V. Realization of ZK-MultNNP

We realize ZK-MultNNP following the three-square decomposition paradigm and Σ-protocol framework of [4]. The novelty lies not in the decomposition method itself but in adapting it to realize the multiplicative non-negativity relation introduced in Section IV.

In the ZKRP construction, the prover demonstrates that $4x(B - x) + 1$ is a sum of three integer squares, thereby proving that $x \in [0, B]$. In ZK-MultNNP, the prover instead demonstrates that $4w_1 w_2 + 1$ is a sum of three integer squares, thereby proving that $w_1 w_2 \ge 0$. The complete construction appears as Protocol 3 in Supplemental Material C. As the proof structure closely follows CKLR, the arguments for completeness and HVZK carry over directly, while the special soundness argument requires extracting two values (w_1, w_2) rather than one, leading to a more involved extraction. We formalize these security guarantees in the following theorem, whose proof appears in Supplemental Material D.

Theorem 1 (Security of Protocol 3): Under Assumption 1, let the verifier sample its challenge uniformly from $\{0, \ldots, C\}$, and let the masking parameters L and S be chosen as in [4, Section 5.3]. Then Protocol 3 is a Σ-protocol for the relation $\mathcal{R}_{\texttt{MultNNP}}$, satisfying the following properties:

- **Completeness:** For any valid witness, an honest execution is accepted with probability 1.
- **Special soundness:** Given two accepting transcripts with identical first message and distinct challenges, one can efficiently extract a valid witness $(w_1, w_2, r_{w_1}, r_{w_2}) \in \mathcal{R}_{\texttt{MultNNP}}$.
- **Honest-verifier zero-knowledge:** Protocol 3 is HVZK with simulation error $9/L$. In particular, if $S = p - 1$, the protocol achieves HVZK against unbounded adversaries.

Having established a concrete realization of ZK-MultNNP, we next combine it with a chain of verified commitments to construct Persephone, our protocol for SPRV.

Protocol 1 Our Solution PERSEPHONE

Parameter: Security parameter λ, Upper bound parameters: random mask bound $S = 2^{2\lambda}$, challenge bound $C = 2^{\lambda}$, and additional security padding parameter $L = 2^{\lambda}$, system upper bound B, Group parameters: group $\mathbb{G}$ with two generators $\mathfrak{g}$ and $\mathfrak{h}$, $pp \triangleq (\mathbb{G}, \mathfrak{g}, \mathfrak{h}, S = 2^{2\lambda})$.

Input: Initial bounds l_0 and u_0, and a sequence of T values $x_1, \cdots, x_T$ for the prover. No input for the verifier.

Output: If a verification fails, the protocol aborts.

Chain Initialization

Prover		**Verifier**
1: Compute $u_0 - l_0$, $B - l_0$ and $B - u_0$		
2: Randomly sample r_{l_0}, $r_{u_0-l_0} \in [0, S]$		
3: Compute $\mathsf{c}_{l_0} = \mathtt{Com.Commit}(pp, l_0; r_{l_0})$		
4: Compute $\mathsf{c}_{u_0-l_0} = \mathtt{Com.Commit}(pp, u_0 - l_0; r_{u_0-l_0})$		
5: Compute $r_{B-l_0} = -r_{l_0}$ and $r_{B-u_0} = -r_{u_0}$		
	$\mathsf{c}_{l_0}, \mathsf{c}_{u_0-l_0}$ $\longrightarrow$	
		6: Compute $\mathsf{c}_{B-l_0} = \mathfrak{g}^B \cdot (\mathsf{c}_{l_0})^{-1}$
		7: Compute $\mathsf{c}_{B-u_0} = \mathfrak{g}^B \cdot (\mathsf{c}_{l_0} \cdot \mathsf{c}_{u_0-l_0})^{-1}$
		8: $\Pi.\mathtt{MultNNP}((\mathsf{c}_{l_0}, \mathsf{c}_{B-l_0}); (l_0, B-l_0, r_{l_0}, r_{B-l_0}))$. If the verification fails, abort.
		9: $\Pi.\mathtt{MultNNP}((\mathsf{c}_{u_0-l_0}, \mathsf{c}_{B-u_0}); (u_0-l_0, B-u_0, r_{u_0-l_0}, r_{B-u_0}))$. If the verification fails, abort.
		10: Store $\overline{\mathsf{c}_{l_0}}$ and $\overline{\mathsf{c}_{u_0}}$

VerifyRanges

Prover		**Verifier**
For $t \in \{1, \cdots, T\}$, do		
1: Randomly sample $r_{x_t} \in [0, S]$		
2: Compute $\mathsf{c}_{x_t} = \mathtt{Com.Commit}(x_t, r_{x_t})$		
3: Compute $y = x_t - l_{t-1}$ and $z = u_{t-1} - x_t$		
4: Compute $r_y = r_{x_t} - r_{l_{t-1}}$ and $r_z = r_{u_{t-1}} - r_{x_t}$		
	c_{x_t} $\longrightarrow$	
		5: Retrieve $\overline{\mathsf{c}_{l_{t-1}}}$ and $\overline{\mathsf{c}_{u_{t-1}}}$
		6: Compute $\mathsf{c}_y = \mathsf{c}_{x_t} \cdot (\mathsf{c}_{l_{t-1}})^{-1}$ and $\mathsf{c}_z = \mathsf{c}_{u_{t-1}} \cdot (\mathsf{c}_{x_t})^{-1}$
		7: $\Pi.\mathtt{MultNNP}((\mathsf{c}_y, \mathsf{c}_z); (y, z, r_y, r_z))$. If the verification fails, the iteration ends: $(c_{l_t}, c_{u_t}) = (c_{l_{t-1}}, c_{u_{t-1}})$ for the verifier, and $(l_t, u_t, r_{l_t}, r_{u_t}) = (l_{t-1}, u_{t-1}, r_{l_{t-1}}, r_{u_{t-1}})$ for the prover.
8: Compute $(l_t, u_t) = f_{update}(l_{t-1}, x_t, u_{t-1})$		
9: Compute $(r_{l_t}, r_{u_t}) = f_{update}(r_{l_{t-1}}, r_{x_t}, r_{u_{t-1}})$		
		10: Compute $(\mathsf{c}_{l_t}, \mathsf{c}_{u_t}) = g_{update}(\mathsf{c}_{l_{t-1}}, \mathsf{c}_{x_t}, \mathsf{c}_{u_{t-1}})$
11: Randomly generate r'_{l_t}, $r'_{u_t} \in [0, S]$		
12: Compute $\mathsf{c}'_{l_t} = \mathtt{Com.Commit}(pp, l_t; r'_{l_t})$		
13: Compute $\mathsf{c}_{u_t} = \mathtt{Com.Commit}(pp, u_t; r'_{u_t})$		
	$\mathsf{c}'_{l_t}, \mathsf{c}'_{u_t}$ $\longrightarrow$	
		14: $\mathtt{Com.Refresh}((pp, \mathsf{c}'_{l_t}, \mathsf{c}_{l_t}); (l_t, r'_{l_t}, r_{l_t}))$. If the verification fails, the iteration ends: $(c_{l_t}, c_{u_t}) = (c_{l_{t-1}}, c_{u_{t-1}})$ for the verifier, and $(l_t, u_t, r_{l_t}, r_{u_t}) = (l_{t-1}, u_{t-1}, r_{l_{t-1}}, r_{u_{t-1}})$ for the prover.
		15: $\mathtt{Com.Refresh}((pp, \mathsf{c}'_{u_t}, \mathsf{c}_{u_t}); (u_t, r'_{u_t}, r_{u_t}))$. If the verification fails, the iteration ends: $(c_{l_t}, c_{u_t}) = (c_{l_{t-1}}, c_{u_{t-1}})$ for the verifier, and $(l_t, u_t, r_{l_t}, r_{u_t}) = (l_{t-1}, u_{t-1}, r_{l_{t-1}}, r_{u_{t-1}})$ for the prover.
		16: Update commitments $c_{l_t} \leftarrow c'_{l_t}$ and $c_{u_t} \leftarrow c'_{u_t}$
		17: Store $\overline{\mathsf{c}_{l_t}}$ and $\overline{\mathsf{c}_{u_t}}$

VI. Persephone

We introduce Persephone, a solution to the problem described in Section III that uses ZK-MultNNP from Section V to reduce the number of proof instances generated and verified at each step after initialization. The protocol achieves this reduction through a *chain* in which each proof reuses bound commitments verified at the preceding step, thereby linking the state across steps. An initialization step verifies the validity of the initial bounds, reducing the verification required at each subsequent step.

Our protocol Π consists of two procedures: `ChainInitialization` and `VerifyRanges`.

At initialization, $t = 0$, the prover proves that its initial lower and upper bounds l_0 and u_0 satisfy $0 \le l_0 \le u_0 \le B$, Equation (5). The prover computes its commitments c_{l_0} and $\mathsf{c}_{u_0-l_0}$ with corresponding randomness r_{l_0} and $r_{u_0-l_0}$. The Π.`ChainInitialization` procedure then invokes two instances of ZK-MultNNP to verify the two conditions in Equation (13). To perform these verifications, the verifier uses c_{l_0}, $\mathsf{c}_{u_0-l_0}$, c_{B-l_0}, and c_{B-u_0}. To reduce communication, the prover sends only the first two commitments, and the verifier computes the remaining two using the additive homomorphic property of Pedersen commitments. If both proofs are accepted, the verifier is assured that the four commitments are consistent with values satisfying the initial bound relation. Protocol 1 specifies the chain initialization Π.`ChainInitialization`.

At each step $t \ge 1$, the verifier begins with commitments to l_{t-1} and u_{t-1} that were verified at the preceding step and whose values satisfy $0 \le l_{t-1} \le u_{t-1} \le B$. Because the invariant already establishes the validity of the bounds, it remains only to prove that $l_{t-1} \le x_t \le u_{t-1}$. By the product equivalence established in Section IV, this condition can be verified using a single ZK-MultNNP instance at each step.

Upon successful verification, the prover updates its bounds locally according to the update rule f_{update} in Equation (4). We can also leverage the homomorphic property of the Pedersen commitment scheme to create a function g_{update} that creates the commitments of the new bounds out of the verified ones of the former bounds and value, $\overline{\mathsf{c}_{l_{t-1}}}$, $\overline{\mathsf{c}_{x_t}}$, and $\overline{\mathsf{c}_{u_{t-1}}}$. Like f_{update}, the function g_{update} is public and reveals neither the bounds nor the current value. The resulting commitments are already verified. Furthermore, the prover can compute the randomness of the new bound commitments locally by applying the same linear update rule to the randomness of the previous bound commitments and the current value commitment.

After updating the bounds and their commitments, we refresh the randomness of their commitments. We use `Com.Refresh` to verify that two commitments refer to the same value: a claimed commitment c'_x of x and a verified commitment $\overline{\mathsf{c}_x}$. At each step, the protocol invokes `Com.Refresh` once for each bound.

The chain maintains the following invariant. At the beginning of step t, the verifier holds $\overline{\mathsf{c}_{l_{t-1}}}$ and $\overline{\mathsf{c}_{u_{t-1}}}$ to bounds satisfying $0 \le l_{t-1} \le u_{t-1} \le B$. At the end of the step, the verifier holds $\overline{\mathsf{c}_{l_t}}$ and $\overline{\mathsf{c}_{u_t}}$ to the resulting bounds, which satisfy $0 \le l_t \le u_t \le B$.

With our solution, we no longer need to assume that the prover uses the updated bounds of Equation (4) in the proof of Equation (3). Indeed, linking commitments across steps prevents this. Oppositely, in the baseline, the prover can use bounds of its choosing as only Equation (3) is verified.

We now state the security guarantees of Persephone, whose proof appears in Supplemental Material B.

Theorem 2 (Security of Persephone*):* Under Assumption 1 (no wrap-around), and assuming that

(i) the Pedersen commitment scheme is binding and hiding,
(ii) ZK-MultNNP is complete, sound, and honest-verifier zero-knowledge for the relation $\mathcal{R}_{\texttt{MultNNP}}$,
(iii) the refresh procedure is complete, sound, and honest-verifier zero-knowledge,

Persephone satisfies completeness, soundness, and honest-verifier zero-knowledge with respect to the private bounds $\{(l_t, u_t)\}_{t=0}^{T}$ and private values $\{x_t\}_{t=1}^{T}$.

In the baseline ZKRP-based approach, each verification step produces an independent proof that a committed value lies within a valid range. Since these proofs share no cryptographic state, nothing inherently enforces that the bounds used in one step are consistent with those verified in the previous step. In contrast, Persephone links each verification step to the previously verified state through a chain of verified commitments. The refresh mechanism ensures that the bounds used at step t are derived from the same values that were verified at step $t-1$. By Theorem 2, this guarantees that the committed state evolves consistently throughout the entire protocol execution, not just within individual steps.

The Refresh protocol can be instantiated using Construction 2.2 of [20], a Σ-protocol for proving that two Pedersen commitments contain the same value.

Persephone uses interactive proofs for both ZK-MultNNP and the Refresh procedure. A non-interactive variant can, in principle, be obtained by applying the Fiat–Shamir transform [23] in the random oracle model. However, in our setting, this requires careful handling of state across iterations, as the challenges must be derived in a way that preserves consistency between proof generation and verification along the chain of commitments. In particular, a naive application of Fiat-Shamir may produce inconsistent transcripts if challenges are not uniquely bound to the current state in the commitment chain. For this reason, our evaluation and security analysis consider only the interactive protocol. We leave a rigorous construction and analysis of a noninteractive variant for future work.

We also note that, while we presented a concrete realization of ZK-MultNNP based on CKLR-style constructions in Section V, the advantage of Persephone is not tied to this specific instantiation. In particular, even when realizing the multiplicative non-negativity relation proof using standard range proofs (e.g., via two calls of existing ZKRP protocols), Persephone still reduces the number of proof instances per verification step compared to the baseline, and thus retains a performance advantage,

albeit smaller. This shows that the gains of Persephone stem from its protocol-level design and are independent of the choice of the underlying proof system.

VII. Protocol Complexity Analysis

	Number of	Baseline	Persephone	
			$t = 0$	$t \geq 1$
Verifier computation complexity	Group inverses	20	9	8
	Group multiplications	58	28	25
	Hash function calls	4	2	1
	Group exponents†	$40b + 164\lambda + 16$	$19b + 82\lambda + 8$	$17b + 49\lambda + 6$
	Random number generations∗	4λ	2λ	λ
Prover computation complexity	Group inverses	12	6	3
	Group multiplications	47	24	14
	Hash function calls	4	2	1
	Group exponents†	$51b + 174\lambda + 16$	$26b + 88\lambda + 8$	$15b + 54\lambda + 6$
	Random number generations∗	$28b + 174\lambda + 24$	$10b + 72\lambda + 2$	$5b + 42\lambda + 4$
	Three square decompositions	4	2	1
Verifier storage (not accumulative)	Bits	0	2	
	Group elements	0	2	
Prover storage (not accumulative)	Number of bits	$2b$	$2b + 4\lambda$	
Verifier communication complexity	Bits	4λ	2λ	3λ
Prover communication complexity	Bits	$20b + 120\lambda + 8$	$10b + 60\lambda + 4$	$5b + 36\lambda + 4$
	Group elements	15	8	8

† Total bit length of the exponent
∗ Total random bit to be generated

TABLE I
Complexity summary for our solution and the baseline at each timestamp (or initialization)

In this section, we compare the complexity of Persephone with that of the ZKRP-based baseline. Both solutions use the Pedersen commitment scheme (see Supplemental Material E for ZKRP and Section V for ZK-MultNNP).

A. Setup and Assumptions

We recall that λ is the security parameter and define $b \triangleq \lceil \log B \rceil$. We assume that all values (l_t, x_t, u_t, and those in f_{update}) are at most b bits, and that integer computations are negligible compared to group operations.

The full complexity analysis is provided in Supplemental Material F (Tables VI, VII, and VIII). We summarize the comparison in Table I under two simplifying assumptions. First, in both solutions, the three-square decomposition is always called with a $(2b + 2)$-bit input and the hash function with five group elements. Because each call has the same worst case cost in both solutions, we compare only the number of calls. Second, since random number generation and group exponentiation (via square-and-multiply) have cost proportional to the number of bits involved, we compare the total number of bits to generate and the total number of exponent bits, respectively.

B. Comparison with ZKRP-based Baseline

Table I summarizes the complexity of a single verification of Equation (3) for both solutions. Our solution incurs a one-time initialization cost that amortizes over T verifications, while the baseline has no initialization. We report asymptotic improvement as the ratio of per-step baseline cost to per-step cost of our solution. We now discuss each complexity aspect in detail.

Verifier computation complexity. Table I shows that our solution requires fewer calls for each building block listed in Table V. This holds for any number of verifications T, including $T = 1$ where the complexity of the initialization is fully passed down to a single verification of Equation (3). For each subsequent step, our solution eliminates 3λ random bits generation, 12 group inverse calls, 33 group multiplication calls, 3 hash function calls, and reduces the total number of bits for group exponents by $23b + 115\lambda + 10$. Asymptotically, the verifier complexity of Persephone is at least 2.32x lower than that of the baseline.

Prover computation complexity. Similar reductions hold for the prover. At each step, our solution eliminates 9 group inverse calls, 33 group multiplication calls, 3 hash function calls, 3 three-square decomposition calls, $23b + 132\lambda + 20$ random bits generation, and reduces the total number of bits for the group exponents by $36b + 120\lambda + 10$. Asymptotically, Persephone reduces prover computation by a factor of at least 2.67 relative to the baseline.

Verifier and prover storage. When proceeding to the next step, stored values from previous steps are no longer required and do not need to be retained. Consequently, storage does not accumulate. The baseline only requires the prover to store the lower and upper bounds l_t and u_t. Hence, the prover's storage is $2b$ bits at each step and no verifier storage is required. On the other

hand, with our solution, the verifier storage stores the verified commitments of the lower and upper bounds l_t and u_t, and the prover stores their randomness and plaintext values l_t and u_t. Hence, Persephone takes $2\mathbb{G}+2$ of verifier storage and $2b+4\lambda$-bit of prover storage. At each step, Persephone requires an additional $2\mathbb{G}+2$ of verifier storage and 4λ-bit of prover storage. We consider this increase in prover and verifier storage as a small trade-off when compared to the reduction in both computation and communication complexity provided by Persephone. We note that Persephone does not eliminate the need for maintaining a full transaction history in a real-world deployment, which is typically required for auditing and compliance. However, the verification protocol itself only requires storing the most recent pair of commitments per user, making the additional storage overhead constant per user.

Verifier communication complexity. With the baseline, the verifier sends one challenge for each of the four ZKRP calls at each step, which gives it a communication complexity of 4λ. Instead, with Persephone, the verifier only needs to send a challenge for a ZK-MultNNP call or a commitment refresh, which gives it a communication complexity of 2λ at initialization and 3λ subsequently. As verified commitments can be retained locally by the verifier, they increase the verifier's storage, but do not affect its communication complexity. Therefore, in contrast to the cumulative computation complexity for which Persephone outperforms the baseline at any step t, the verifier cumulative communication complexity is strictly better only from $t = 3$ onwards. At $t = 1$, the cumulative communication complexity for the verifier is λ-bit higher for our solution, while at $t = 2$, it is identical for both solutions. For subsequent steps, our solution sends λ-bit fewer than the baseline at each step t. Asymptotically, the verifier communicates 1.33x fewer elements with Persephone.

Prover communication complexity. The cumulative prover communication of Persephone is not lower for every t under all group element encodings. At $t = 1$, our solution communicates $5b + 24\lambda$ fewer bits than the baseline but sends one additional group element. At every subsequent step $t \geq 2$, it communicates $15b + 84\lambda + 4$ fewer bits and seven fewer group elements. Consequently, the cumulative prover communication of Persephone is lower for every $t \geq 2$, regardless of the group element encoding. It is also lower at $t = 1$ whenever each group element is represented using fewer than $5b + 24\lambda$ bits. This holds in practice. For example, with $b = \lambda = 32$ and group elements represented as 510-bit strings, the condition is satisfied. Furthermore, asymptotically, the prover communicates at least 1.875x fewer elements with Persephone.

C. Comparison with Bulletproofs-based Baseline

	Proof Size (in bits)	Prover Complexity	Verifier Complexity
Persephone	$8\mathbb{G} + (5b + 36\lambda + 4)$	$10\mathsf{C}_{MSM}(2) + \mathsf{C}_{MSM}(5) + \mathsf{C}_{3SP}(b)$	$2\mathsf{C}_{MSM}(2) + 5\mathsf{C}_{MSM}(3) + \mathsf{C}_{MSM}(6)$
Bulletproofs [11]	$(2\lceil\log b\rceil + 8)\mathbb{G} + 10\lambda$	$2\mathsf{C}_{MSM}(8b+1) + \mathsf{C}_{MSM}(8b) + (\log(b) + 4)\mathsf{C}_{MSM(2)}$	$\mathsf{C}_{MSM}(8b + 8\log(b) + 22)$.
Bulletproofs+ [12]	$(2\lceil\log b\rceil + 7)\mathbb{G} + 6\lambda$	$\mathsf{C}_{MSM}(8b + 2) + \mathsf{C}_{MSM}(8b + 1) + \mathsf{C}_{MSM}(8b - 1) + (2\log b + 4)\mathsf{C}_{MSM}(2)$	$\mathsf{C}_{MSM}(8b + 2\log b + 8)$

TABLE II
Complexity summary for Persephone and Bulletproofs-based baseline and its variants for one transaction (not the initialization step)

To broaden the comparison, we consider two ZKRP baselines instantiated using Bulletproofs [11] and Bulletproofs+ [12], respectively. Since the baseline requires four proofs per step while Persephone requires one, and since a single ZK-MultNNP has comparable cost to a single ZKRP, our approach achieves a constant-factor reduction.

A key advantage of Bulletproofs is the ability to aggregate multiple range proofs into a single proof, reducing proof size and amortizing verification cost. However, in our sequential setting, each verification step depends on the previously verified state, so cross-step aggregation is not possible. Batching can only be applied within a single step, fixing the effective batch size to four. Crucially, aggregation does not reduce the number of statements being proven per step, so it does not eliminate the multiplicative gap in prover complexity between the baseline and Persephone.

For this comparison, we focus on the dominant operations. We consider only multi-scalar multiplications (MSMs) and, for Persephone, the three-square decomposition step. We ignore lower-order costs such as scalar-field operations, hash computations, field multiplications, and field inversions. We denote by $\mathsf{C}_{MSM}(n)$ the cost of computing an MSM with n terms, i.e., $\prod_{i=1}^{n} \mathfrak{g}_i^{e_i}$ for $\mathfrak{g}_i \in \mathbb{G}$, and by $\mathsf{C}_{3SP}(n)$ the cost of the three-square decomposition on an n-bit input. Various algorithms may be used to evaluate each MSM instance, including naive approaches, Straus-style methods, or Pippenger-type algorithms [24].

For Persephone, as described in Table I, the proof size is $8\mathbb{G} + 5b + 36\lambda + 4$ bits. Considering Algorithms 1, 3, and 2, the prover and verifier complexity per verification step are $10\mathsf{C}_{MSM}(2) + \mathsf{C}_{MSM}(5) + \mathsf{C}_{3SP}(b)$ and $2\mathsf{C}_{MSM}(2) + 5\mathsf{C}_{MSM}(3) + \mathsf{C}_{MSM}(6)$, respectively. Notably, the MSM profile of Persephone is independent of the bit length b. Unlike Bulletproofs-style constructions, our integer-based approach does not rely on a recursive inner-product argument whose complexity scales with the witness dimension. The only component depending on b is the three-square decomposition.

An analysis of Bulletproofs [11] and Bulletproofs+ [12] yields the results summarized in Table II. We emphasize that Persephone is presented without implementation-level optimizations, whereas the Bulletproofs baselines include all proposed optimizations from their respective works. For proof size and verifier complexity, the terms in Table II are adapted directly from [11, 12], while prover complexity is estimated from the provided algorithms and proposed optimizations.

Because the verifier complexity of Persephone is independent of b, whereas the verifier complexities of both baselines grow with b, Persephone achieves lower verifier complexity for sufficiently large b. For prover complexity, all components except the three-square decomposition are also independent of B. Hence, assuming an efficient decomposition algorithm (e.g., Rabin-Shallit [25]), Persephone also provides better prover complexity for sufficiently large B.

VIII. Experimental Evaluation of Persephone

In this section, we compare the performance of Persephone with the CKLR-based ZKRP baseline under practical parameters. We first instantiate the theoretical complexity analysis using concrete field operation counts and group element sizes in Section VIII-A. We then report experimental runtimes over $T = 100$ verification steps in Section VIII-B.

A. Practical Complexity

	Number of	Baseline	Persephone		
			$t = 0$	$t \geq 1$	$T = 1$
Verifier computation complexity†	Field Operations	85,754	42,444	27,823	70,267
Prover computation complexity†	Field Operations	94,327	47,792	28,932	76,724
Verifier storage (not accumulated)	Bits	0	1,022		
Prover storage (not accumulated)	Bits	64	192		
Verifier communication complexity	Bits	128	64	96	160
Prover communication complexity	Bits	12,138	6,324	5,396	11,720

† without hash calculation, and three-square decomposition when applicable

TABLE III
Practical complexity for our solution and the baseline at each timestamp (or initialization)

Table III summarizes the complexity analysis of Section VII for a practical verification scenario, with further details below. We consider the security parameter $\lambda = 32$, system upper bound $B = 2^{32}$, and Curve25519 [26] for the group $\mathbb{G}$, which is defined over a prime field $\mathbb{F}_p$ of size $p = 2^{255} - 19$. A group element is a point on this elliptic curve, represented as a pair of field elements. Each group element is encoded as a 510-bit string. Furthermore, point multiplication and inverse operation cost 8 and 9 field multiplications, respectively. Finally, a group exponentiation operation with an ℓ-bit exponent costs 13ℓ field multiplications.

		Baseline		Persephone		Speed-Up
		Mean	St Dev	Mean	St Dev	
Three-square decomposition		1309	1542	12.25	0.3804	106.8
Hash function	Prover	0.1295	0.004941	0.03293	0.002446	3.932
	Verifier	0.1250	0.003991	0.03313	0.003773	3.774
Field operations	Prover	3.730	0.04592	1.14242	0.01567	3.265
	Verifer	3.412	0.03710	1.23462	0.01463	2.764
Total (w\o three-square decomposition)	Prover	3.859	0.04602	1.175	0.0159	3.284
	Verifier	3.542	0.03756	1.269	0.01476	2.791
	Prover + Verifier	7.401	0.07542	2.445	0.02214	3.027

TABLE IV
Amortized time and associated standard deviation (in ms) for the different steps of our solution and the baseline

For computational complexity, Table III reports only the number of field operations, omitting steps involving integer operations, random number generation, three-square decomposition, and hash functions. First, we treat integer operations, such as those used to compute the ZK-MultNNP responses, as negligible relative to field operations. Second, the three-square decomposition, in either solution, always has a $(2b + 2)$-bits input, while the hash function, in either solution, is called with a 5 group elements input, each represented with a 510-bit string. Hence, the comparison based on the number of calls in Section VII still applies, and Persephone reduces the number of calls for each component by a factor of four. Similarly, the baseline generates at least 4.14 times as many random bits as Persephone. Hence, we find a lower bound of our improvement for the computation complexity when considering the number of field operations, which is the focus of this section. Therefore, by combining these observations with the reduced number of field operations summarised in Table III, Persephone achieves asymptotic speedups of at least 3.08x for the prover and 3.26x for the verifier. Moreover, even in the worst-case scenario where the initialization cost is amortised over a single verification ($T = 1$), our solution offers improved computation complexity. After $T = 3$ verifications, Persephone is at least twice as fast as the baseline, and exceeds a 3x speedup for $T = 56$ verifications.

Our solution also achieves a 2.25x improvement in prover communication complexity and 1.33x improvement in verifier communication complexity. However, these gains in communication and computation complexity come at the cost of increased verifier and prover storage. Nonetheless, we consider this a minor limitation in practice given the significant computation and, to a lesser extent, communication complexity benefits offered by Persephone.

B. Experimental Results

We perform our experiments on an AWS EC2 M5 instance equipped with a 2nd Generation Xeon Platinum 8000 processor @ 3.1 GHz, VCPU 16, and 64GB of RAM, with a network bandwidth of 10 Gbps, running on Ubuntu 24.04. We implemented our solution and the baseline in Rust. The source code is publicly available at https://github.com/ant-dtc/persephone-service.

We again use $\lambda = 32$ and set $B = 2^{32}$. We consider initial bounds $l_0 = 300$ and $u_0 = 200,000$. At each step, we sample x_t uniformly from the integers in $[l_{t-1}, \lfloor (l_{t-1} + u_{t-1})/2 \rfloor]$. We define the update function f_{update} by

$$(l_t, u_t) = f_{update}(l_{t-1}, x_t, u_{t-1}) = (x_t - l_{t-1}, u_{t-1} - x_t). \tag{16}$$

By construction, f_{update} satisfies $0 \le l_t \le u_t \le u_0 \le B$.

We instantiate Protocol 3 using Pedersen commitments and instantiate the baseline ZKRP using the construction of [4]. Regarding the three-square decomposition, we use exhaustive search as no better open-source alternative is currently available. While some studies have proposed more efficient methods [25, 27], their implementations are not publicly released, necessitating our use of a less efficient exhaustive search. Given an input w, we search through $y_1 \in [0, \sqrt{w}]$, $y_2 \in \left[0, \sqrt{w - y_1^2}\right]$, and $y_3 \in \left[0, \sqrt{w - y_1^2 - y_2^2}\right]$.

Table IV reports the amortized time per verification step for all operations required to verify Equation (3) over a sequence of $T = 100$ steps. We also report the amortized time for the three-square decomposition, hash function, and field operations step separately. We exclude the three-square decomposition time from the total so that the reported improvement reflects only the gains inherent to Persephone's design and remains independent of the specific decomposition algorithm used. Furthermore, the total includes the time spent on hash evaluations and field operations, together with other components such as integer arithmetic and comparisons.

Three-square decomposition. Persephone performs the decomposition into three squares approximately 106 times faster than the baseline, with significantly lower variance (coefficient of variation of 3% compared to 117% for the baseline). Both solutions use the same decomposition algorithm, so the observed difference arises from the number and magnitudes of the inputs rather than from the underlying primitive.

This gap is explained by two factors, both independent of the decomposition algorithm. First, Persephone reduces the number of decomposition calls per verification step from four to one, Section VII. Second, while the input to the decomposition is $(2b + 2)$ bits long in both solutions, the effective magnitude of the input is smaller in Persephone. In the baseline, the ZKRP input is $4w(B - w) + 1$, where the factor $B - w$ remains of the same order as B even for small w. In Persephone, the input is $4w_1w_2 + 1$ with $w_1 = (x_t - l_{t-1})$ and $w_2 = (u_{t-1} - x_t)$, both of which can be small when x_t is close to its bounds. Together, these two factors yield at least a 4x improvement.

Our implementation uses exhaustive search for the decomposition, as no open-source implementation of more efficient algorithms such as the Rabin–Shallit algorithm [25] is currently available. By a classical result of Gauss [28], valid decompositions are dense ($r_3(N) = \Theta(\sqrt{N})$), so exhaustive search terminates quickly in practice. However, the linear dependence on input magnitude amplifies the gap between the two solutions beyond the structural 4x factor, explaining the observed 106x improvement.

Overall performance. Persephone outperforms the baseline on every component, achieving lower amortized time and lower standard deviation throughout. Overall, Persephone is at least 3x faster than the baseline. Since we exclude the three-square decomposition time from the total, this speedup reflects only the gains inherent to Persephone's design and is independent of the decomposition algorithm used. Including the decomposition time would further increase the reported improvement, making the 3x speedup a conservative lower bound.

IX. Application to Payment Systems

In this section, we evaluate Persephone in the context of a digital payment system. Each user has a wallet whose balance is known only to its owner, and each transfer amount is hidden from all parties except the sender and receiver, as illustrated in Figure 2. Privacy-preserving payments may also need to support regulatory or risk-assessment functions. For example, Yi et al. combine homomorphic encryption with zero knowledge proofs to detect anomalous private transactions while preserving the confidentiality of transaction contents [29]. Persephone is complementary to such approaches, as it focuses on efficiently verifying the validity of sequential private transaction amounts and balances. Persephone augments the payment system with a chain of verified commitments (shown in red) that stores the commitments produced during transaction verification, enabling subsequent transactions to be verified faster.

We measure transaction response time for Persephone and the CKLR-based baseline. A widely adopted responsiveness criterion is the Doherty threshold of 400 ms [5], beyond which users perceive the system as unresponsive and, under sustained load, delays accumulate across concurrent transactions. We consider 100 sequential transactions, each with a value randomly chosen between 0 and 4000. Figure 3 shows the runtime of each transaction. Persephone achieves 270 ± 2 ms, **always** satisfying the Doherty threshold. In contrast, the ZKRP-based baseline takes 6292 ± 9260 ms on average, exceeding the threshold in 70% of cases and requiring on average more than 15x the 400 ms limit. Persephone thus delivers both faster and more predictable

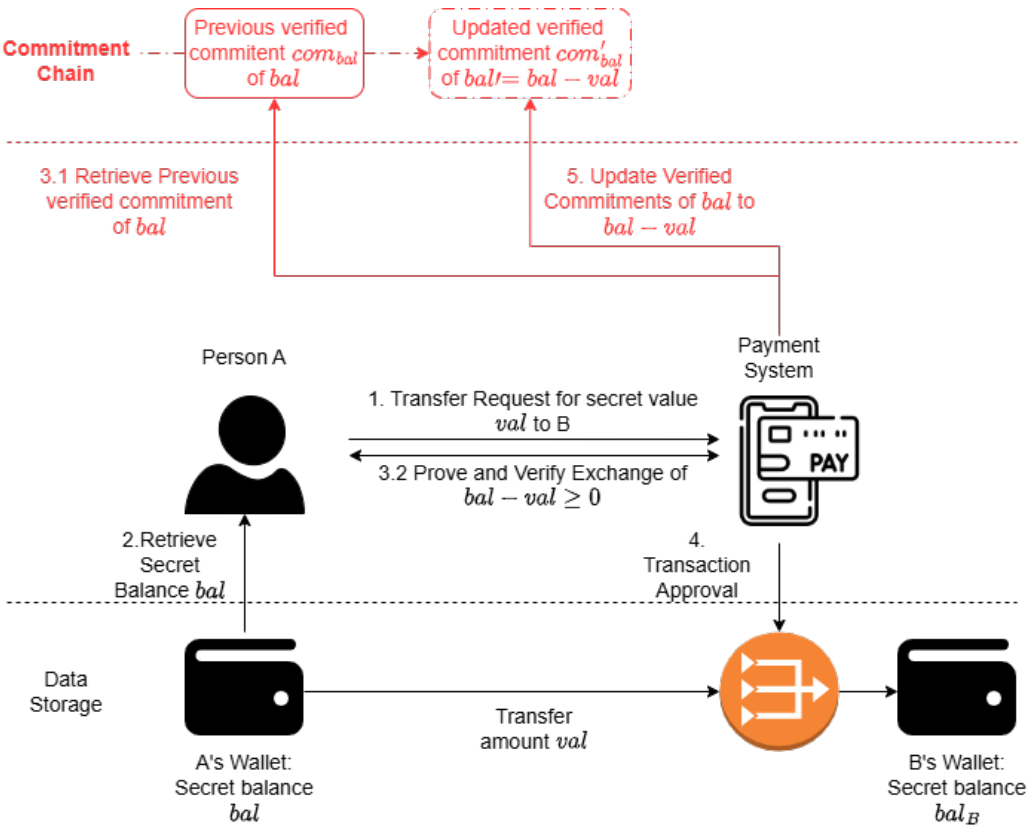

Fig. 2. Integration of Persephone into a blockchain payment system that preserves transaction privacy.

response times. We note that these results reflect ideal conditions, where each transaction is processed immediately without queuing. In a real deployment, where concurrent users compete for system resources, the baseline's performance would degrade further.

Persephone assumes that transactions for a given account are processed sequentially, with each verification step operating on the latest verified state. This aligns with many payment systems where transactions are already serialized to ensure balance consistency [2, 30]. When concurrent transaction requests arise, mechanisms such as application-level serialization, commitment versioning, or independent per-account chains with reconciliation could be employed to maintain consistency. Extending Persephone to natively support concurrent verification remains an interesting direction for future work.

X. Conclusion

In this work, we presented Persephone, a new solution to the sequential private range verification problem with direct applications to digital payment systems, where transactions must be verified without revealing sensitive user information such as balances or transfer amounts. Persephone improves upon the ZKRP-based baseline through two key mechanisms. First, it replaces ZKRP with a novel building block, ZK-MultNNP, reducing the number of zero-knowledge proof invocations per verification. Second, it reuses previously verified commitments to simplify subsequent verifications by forming a chain that links consecutive transactions. Our theoretical analysis and experimental evaluation show that Persephone consistently outperforms ZKRP-based solutions by a factor of at least 3x, a conservative estimate that excludes the additional gains from the three-square decomposition step. We further demonstrated its practicality in a blockchain-based digital payment scenario, where significant efficiency gains at the system level lead to a more stable verification pipeline with minimal cumulative delay. These results position Persephone as a practical building block for privacy-preserving payment systems and, more broadly, for any application requiring efficient sequential private range verification.

References

[1] R. Auer, R. Böhme, J. Clark, and D. Demirag, "Privacy-enhancing technologies for digital payments: mapping the landscape," Tech. Rep. 1242, Bank for International Settlements, 2025.

[2] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *International Conference on Financial Cryptography and Data Security*, 2020.

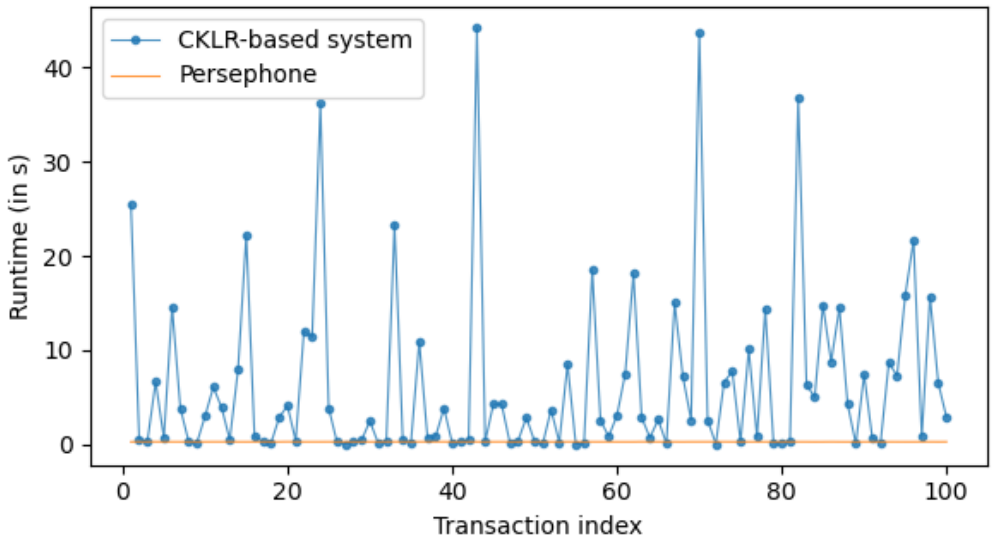

Fig. 3. Runtime per transaction for PERSEPHONE and the CKLR-based solution over the 100 transaction application workload.

[3] G. Maxwell, "Confidential transactions - investigation." https://elementsproject.org/features/confidential-transactions/investigation.
[4] G. Couteau, M. Klooß, H. Lin, and M. Reichle, "Efficient range proofs with transparent setup from bounded integer commitments," in *Annual international conference on the theory and applications of cryptographic techniques*, 2021.
[5] W. J. Doherty and A. J. Thadani, "Designing for instant feedback: The doherty threshold," 1982.
[6] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *IEEE symposium on security and privacy*, 2014.
[7] Z. J. Williamson, "The aztec protocol," 2018.
[8] W. Yong, C. Lijie, W. Yifan, and W. Qiancheng, "Efficient and secure confidential transaction scheme based on commitment and aggregated zero-knowledge proofs," *Journal of Cyber Security Technology*, vol. 8, no. 4, pp. 312–332, 2024.
[9] M. Christ, F. Baldimtsi, K. K. Chalkias, D. Maram, A. Roy, and J. Wang, "Sok: Zero-knowledge range proofs," in *6th Conference on Advances in Financial Technologies*, 2024.
[10] C. Lin, D. He, X. Huang, M. K. Khan, and K.-K. R. Choo, "Dcap: A secure and efficient decentralized conditional anonymous payment system based on blockchain," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2440–2452, 2020.
[11] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *IEEE symposium on security and privacy (SP)*, 2018.
[12] H. Chung, K. Han, C. Ju, M. Kim, and J. H. Seo, "Bulletproofs+: Shorter proofs for a privacy-enhanced distributed ledger," *IEEE Access*, 2022.
[13] L. Eagen, S. Kanjalkar, T. Ruffing, and J. Nick, "Bulletproofs++: Next generation confidential transactions via reciprocal set membership arguments," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 249–279, Springer, 2024.
[14] Y. Zhou and S. C.-K. Chau, "Verange: Verification-efficient zero-knowledge range arguments with transparent setup for blockchain applications and more," *Cryptology ePrint Archive*, 2025.
[15] S. Kim, G. Lee, H. Lee, and J. H. Seo, "Leopard: Sublinear verifier inner product argument under discrete logarithm assumption," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 5332–5344, 2023.
[16] G. Couteau, D. Goudarzi, M. Klooß, and M. Reichle, "Sharp: Short relaxed range proofs," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022.
[17] K. Chalkias, S. Cohen, K. Lewi, F. Moezinia, and Y. Romailler, "HashWires: Hyperefficient credential-based range proofs." Cryptology ePrint Archive, Paper 2021/297, 2021.
[18] W. Li, Z. Zhang, Y. Guo, S. S. M. Chow, and Z. Wan, "Succinct hash-based arbitrary-range proofs," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 145–158, 2025.
[19] D. Mouris and N. G. Tsoutsos, "Zilch: A framework for deploying transparent zero-knowledge proofs," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3269–3284, 2021.
[20] S. Celi, S. Levin, and J. Rowell, "Cdls: Proving knowledge of committed discrete logarithms with soundness," in *International Conference on Cryptology in Africa*, pp. 69–93, Springer, 2024.

[21] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, 1985.
[22] E. Fujisaki and T. Okamoto, "Statistical zero knowledge protocols to prove modular polynomial relations," in *Annual International Cryptology Conference*, pp. 16–30, Springer, 1997.
[23] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*, 1986.
[24] D. J. Bernstein, "Pippenger's exponentiation algorithm," *Preprint*, 2002.
[25] M. O. Rabin and J. O. Shallit, "Randomized algorithms in number theory," *Communications on Pure and Applied Mathematics*, 1986.
[26] D. J. Bernstein, "Curve25519: New diffie-hellman speed records," in *Public Key Cryptography*, 2006.
[27] P. Pollack and P. Schorn, "Dirichlet's proof of the three-square theorem: An algorithmic perspective," *Mathematics of Computation*, 2019.
[28] G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers*. Oxford university press, 1979.
[29] L. Yi, J. Liu, Z. Wan, K. Ren, and C. Chen, "Regulatable and privacy-preserving blockchain via anomaly detection on private transactions," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 12126–12140, 2025.
[30] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014.
[31] J. Groth, "Non-interactive zero-knowledge arguments for voting," in *International Conference on Applied Cryptography and Network Security*, pp. 467–482, Springer, 2005.

APPENDIX

A. Commitment Refresh for Pedersen Commitment

In Protocol 2, we briefly describe a commitment refresh protocol for the Pedersen commitment scheme, as described in [20, Construction 2.2]. Intuitively, given that two commitments are of the same value, their ratio is a power of $\mathfrak{h}$. Hence, in the protocol, the prover tries to convince the verifier of its knowledge of the exponent of the ratio when expressed as a power of $\mathfrak{h}$.

Protocol 2 `Com.Refresh` with the Pedersen commitment scheme

Parameters: Security parameter λ with random mask bound $S = 2^{2\lambda}$ and additional security padding parameter $L = 2^{\lambda}$, second group generator $\mathfrak{h}$ of Pedersen commitment.
Inputs: commitment c with randomness r, commitment c' with randomness r'.
Output: If c and c' are commitments of the same value, the verifier accepts with probability 1.

Prover		**Verifier**
1: Randomly sample $\alpha \in [0, 2SL]$.		
	$\mathsf{t} = \mathfrak{h}^{\alpha}$ $\longrightarrow$	
		2: Randomly sample $\beta \in [0, L]$
	$\longleftarrow$ β	
	$\gamma = \beta(r - r') + \alpha$ $\longrightarrow$	
		3: **if** $\mathfrak{h}^{\gamma} == \mathsf{t}(\mathsf{c}/\mathsf{c}')^{\beta}$ **then**
		4: Return 1 ▷ the verification is successful, the two commitments are of the same value.
		5: **else**
		6: Return 0 ▷ the verification failed

B. Proof of Theorem 2

PROOF. We prove completeness, soundness, and honest-verifier zero-knowledge separately.

Completeness. Assume that for every iteration $t \in \{1, \ldots, T\}$, Equation (3) holds, the prover follows the protocol honestly, and the prover's locally updated bounds (l_t, u_t) are computed according to the prescribed update rule $f_{\mathsf{update}}(l_{t-1}, x_t, u_{t-1})$. We show by induction on t that the verifier accepts at every iteration.

For the initialization step, Protocol 1 invokes two instances of ZK-MultNNP to prove the two multiplicative non-negative conditions corresponding to the validity of the initial bounds, i.e., $l_0(B - l_0) \geq 0$ and $(u_0 - l_0)(B - u_0) \geq 0$. Since these statements are true and ZK-MultNNP is complete by assumption, both proofs are accepted. Hence the verifier stores valid commitments to the initial bounds.

Now assume that at the beginning of iteration $t \in \{1, \ldots, T\}$ the verifier stores valid commitments to (l_{t-1}, u_{t-1}). If Equation (3) holds, then

$$l_{t-1} \leq x_t \leq u_{t-1},$$

which is equivalent to

$$(u_{t-1} - x_t)(x_t - l_{t-1}) \geq 0$$

by the equivalence established in Section IV. Therefore the ZK-MultNNP instance used in iteration t is invoked on a true statement and is accepted by completeness of ZK-MultNNP. The verifier then computes the updated commitments homomorphically using the public update rule, while the prover computes the corresponding updated plaintext bounds locally. Since the prover follows the update rule honestly, both sides refer to the same updated values. This allows the refresh checks to succeed by correctness of the refresh procedure. Thus the verifier accepts iteration t and ends the iteration with valid refreshed commitments to (l_t, u_t). By induction, the verifier accepts throughout the execution.

Soundness. First, suppose that the relation $0 \leq l_0 \leq u_0 \leq B$ is not satisfied. Then by the soundness of the underlying ZK-MultNNP instances, the verifier rejects except with negligible probability.

Now we show soundness for iteration $t \in \{1, \ldots, T\}$. We prove this by induction on t. More specifically, we establish the following two claims.

- At the beginning of iteration t, the verifier stores two verified commitments $(\overline{c_{l_{t-1}}}, \overline{c_{u_{t-1}}})$ that, except with negligible probability, correspond to the prover's local bounds (l_{t-1}, u_{t-1}) where

$$\begin{cases} f_{\mathsf{Update}}(l_{t-2}, x_{t-1}, u_{t-2}), & \text{if iteration } t-1 \text{ is accepted,} \\ (l_{t-2}, u_{t-2}), & \text{if iteration } t-1 \text{ is rejected} \end{cases}.$$

- Given these verified commitments, if either (l_{t-1}, x_t, u_{t-1}) does not satisfy Equation (3), or $(l_t, u_t) \neq f_{\mathsf{Update}}(l_{t-1}, x_t, u_{t-1})$ at the end of the iteration, then verifier rejects except with a negligible probability.

We establish the following implications between the two claims.

- For $t \geq 1$, if the first claim holds at iteration t, then the second claim also holds at iteration t. If (l_{t-1}, x_t, u_{t-1}) does not satisfy Equation (3), then by the soundness of the underlying ZK-MultNNP, the verifier rejects except with a negligible probability.
  Otherwise, suppose Equation (3) is satisfied. In this case, acceptance additionally requires that the final commitments (c_{l_t}, c_{u_t}) at the end of iteration t to be valid commitments of the correctly updated (l_t, u_t).
  The verifier holds commitments $(\overline{c_{l_t}}, \overline{c_{u_t}})$ resulting from the previous steps of the protocol. These commitments are consistent with the prover's state provided that all prior checks have been passed.
  By the soundness of the refresh protocol, given the refreshed commitments (c'_{l_t}, c'_{u_t}) produced by the prover, if these do not correspond to the same underlying values as $(\overline{c_{l_t}}, \overline{c_{u_t}})$, then the verifier rejects except with negligible probability. Hence, the refresh procedure enforces consistency of the committed values across iterations.
  Acceptance of the ZK-MultNNP instance enforces that (l_{t-1}, x_t, u_{t-1}) satisfies Equation (3). The correctness of the updated bounds is ensured by the consistency between the verifier's homomorphic update and the refresh procedure.
- For $t \geq 2$, if the second claim holds at iteration $t-1$, then the first claim holds at iteration t. Indeed, at the end of iteration $t-1$, if it is accepted, then by the second claim, the updated bounds are the correctly computed $(l_{t-1}, u_{t-1}) = f_{\mathsf{Update}}(l_{t-2}, x_{t-1}, u_{t-2})$ and the verifier stores verified commitments of such values. Otherwise, no state update is done and we have $(l_{t-1}, u_{t-1}) = (l_{t-2}, u_{t-2})$. This means that the first claim holds at iteration t.

Hence, to show that both claims are satisfied in all iterations $t \geq 1$, it is sufficient to show that the first claim holds in iteration $t = 1$.

At the beginning of iteration $t = 1$, the commitments are those verified at iteration $t = 0$. As argued above, Persephone is sound at iteration $t = 0$, implying that if the protocol does not abort, then except with a negligible probability, $0 \leq l_0 \leq u_0 \leq B$ and the verifier holds the verified commitments of the initial bounds. This shows the correctness of the first claim in iteration $t = 1$, completing the proof of soundness.

Honest-verifier zero-knowledge. We construct a simulator $\mathcal{S}$ for the verifier's view over the full execution. At each iteration, the verifier's view consists of: commitments sent by the prover, the transcript of the invoked ZK-MultNNP proof, and the transcript of the refresh procedure.

Recall that due to the assumption that the underlying ZK-MultNNP and the refresh procedure have the property of honest verifier zero knowledge, we can assume the existence of the simulators $\mathcal{S}_{\mathsf{ZK-MultNNP}}$ and $\mathcal{S}_{\mathsf{Refresh}}$. The simulator $\mathcal{S}$ can then be constructed as follows. It first randomly samples (c_l, c_u) as valid Pedersen commitments using fresh randomness. The hiding property of the Pedersen commitment guarantees that the pair of commitments cannot be distinguished from the commitments of the real values of l_0 and u_0. Next, for each time ZK-MultNNP is invoked, $\mathcal{S}$ calls $\mathcal{S}_{\mathsf{ZK-MultNNP}}$ to simulate the ZK-MultNNP transcript given only the public inputs and the corresponding accept/reject outcome. Lastly, for each time Refresh is invoked, $\mathcal{S}$ calls $\mathcal{S}_{\mathsf{Refresh}}$ to simulate the refresh transcripts analogously.

By our assumptions on the underlying building blocks, from the verifier's point of view, the interaction with $\mathcal{S}$ is computationally indistinguishable from that of Persephone.

Hence Persephone is honest-verifier zero-knowledge with respect to the sequence of private values $\{(l_t, x_t, u_t)\}_{t=0}^{T}$. □

C. Realization of ZK-MultNNP

We provide in Protocol 3 our realization of ZK-MultNNP.

D. Proof of Theorem 1

Proof. We prove completeness, special soundness, and honest-verifier zero-knowledge separately.

Completeness. Completeness follows directly from the correctness of the three-square decomposition. Given valid witnesses (w_1, w_2) such that $w_1 w_2 \geq 0$, by [25, 31], there exist integers y_1, y_2, y_3 such that

$$4w_1w_2 + 1 = y_1^2 + y_2^2 + y_3^2.$$

The prover then computes the corresponding commitments and responses according to the protocol.

Protocol 3 Realization of ZK-MULTNNP

Parameters: Security parameter λ, Upper bound parameters: random mask bound $S = 2^{2\lambda}$, challenge bound $C = 2^{\lambda}$, and additional security padding parameter $L = 2^{\lambda}$.

Inputs: Two public commitments c_{w_1} and c_{w_2} of two inputs w_1 and w_2 with associated random masks r_{w_1} and r_{w_2}. Inputs and private random masks are held by the prover, while public inputs are known by both prover and verifier.

Output: If $w_1 w_2 \geq 0$, the verifier accepts with probability 1.

Initialization

Prover

1: Compute y_1, y_2, and y_3 with bit length $(2+\lceil\log(w_1 w_2)\rceil)/2$ s.t. $(y_1, y_2, y_3) = \mathcal{F}^{(3\mathsf{SP})}(4w_1 w_2 + 1)$
2: For $i \in \{1,2,3\}$, randomly sample $r_{y_i} \in [0, S]$
3: For $i \in \{1,2,3\}$, compute $\mathsf{c}_{y_i} = \mathfrak{g}^{y_i}\mathfrak{h}^{r_{y_i}}$
4: For $i \in \{0,1,2,3\}$, randomly sample $m_i \in [0, BCL]$
5: For $i \in \{0,1,2,3\}$, randomly sample $s_i \in [0, SCL]$
6: For $i \in \{0,1,2,3\}$, compute $\mathfrak{d}_i = \mathfrak{g}^{m_i}\mathfrak{h}^{s_i}$
7: Randomly sample $\sigma \in [0, 4SBCL]$
8: Compute $\mathfrak{d} = \mathfrak{h}^{\sigma}\left(\mathsf{c}_{w_2}\right)^{4m_0}\mathsf{c}_{y_1}^{-m_1}\mathsf{c}_{y_2}^{-m_2}\mathsf{c}_{y_3}^{-m_3}$
9: Compute $\Delta = \mathbb{H}(\mathfrak{d}_0\|\mathfrak{d}_1\|\mathfrak{d}_2\|\mathfrak{d}_3\|\mathfrak{d})$

$\mathsf{c}_{y_1}, \mathsf{c}_{y_2}, \mathsf{c}_{y_3}, \Delta \longrightarrow$

Challenge

Prover **Verifier**

1: Randomly sample $\gamma \in [0, C]$

$\leftarrow \gamma$

Response

Prover

1: For $i \in \{0,1,2,3\}$, compute $z_i = m_i + \gamma y_i$ with $y_0 = w_1$ and $t_i = s_i + \gamma r_{y_i}$ with $r_{y_0} = r_{w_1}$
2: Compute $\tau = \sigma + \gamma\left(\sum_{i=1}^{3} y_i r_{y_i} - 4r_{w_2} w_1\right)$

$z_0, z_1, z_2, z_3, t_0, t_1, t_2, t_3, \tau \longrightarrow$

Verification

Prover

Verifier

1: **for** $i \in \{1,2,3\}$ **do**
2: **if** $z_i \notin [0, 2BCL]$ **then**
3: Return 0 ▷ the verification failed, z_i is not sufficiently small.
4: **for** $i \in \{0,1,2,3\}$ **do**
5: Compute $\mathfrak{f}_i = \mathfrak{g}^{z_i}\mathfrak{h}^{t_i}\mathsf{c}_{y_i}^{-\gamma}$ with $\mathsf{c}_{y_0} = \mathsf{c}_{w_1}$
6: Compute $\mathfrak{f} = \mathfrak{h}^{\tau}\mathfrak{g}^{\gamma}\left(\mathsf{c}_{w_2}\right)^{4z_0}\prod_{i=1}^{3}\mathsf{c}_{y_i}^{-z_i}$
7: **if** $\Delta = \mathbb{H}(\mathfrak{f}_0\|\mathfrak{f}_1\|\mathfrak{f}_2\|\mathfrak{f}_3\|\mathfrak{f})$ **then**
8: Return 1 ▷ the verification is successful, c_{w_i} is a valid commitment of w_i, $\mathfrak{f}_i = \mathfrak{d}_i$, and $4w_1 w_2 + 1 = y_1^2 + y_2^2 + y_3^2$, $\mathfrak{f} = \mathfrak{d}$.
9: **else**
10: Return 0 ▷ the verification failed

Assuming the prover is honest, direct substitution shows that the verification equations satisfy

$$f_i = d_i \quad \text{for } i \in \{0,1,2,3\}, \quad \text{and} \quad f = d.$$

Furthermore, all range constraints are satisfied by construction of the protocol parameters. Hence, the recomputed hash value matches the original value Δ, and the verifier accepts with probability 1.

Special soundness. Next, similar to [4, Section 5], we prove that Protocol 3 satisfies 2-special soundness.

That is, given two accepting transcripts with the same $(\mathsf{c}_{y_1}, \mathsf{c}_{y_2}, \mathsf{c}_{y_3}, \Delta)$ and distinct challenges $\gamma^{(1)} \neq \gamma^{(2)}$, with corresponding responses

$$(\gamma^{(i)}, z_0^{(i)}, z_1^{(i)}, z_2^{(i)}, z_3^{(i)}, t_0^{(i)}, t_1^{(i)}, t_2^{(i)}, t_3^{(i)}, \tau^{(i)}) \text{ for } i = 1,2,$$

we show that one can efficiently extract a witness $(w_1', w_2', r_{w_1'}, r_{w_2'})$ such that $\mathsf{c}_{w_i} = g^{w_i'} h^{r_{w_i'}}$ for $i = 1,2$, and $(w_1', w_2', r_{w_1'}, r_{w_2'}) \in \mathcal{R}_{\mathsf{MultNNP}}$.

For simplicity of notation, for any variable described in Protocol 3, we add the superscript (i) to indicate that it is the value of such variable in the i-th execution of the protocol for $i = 1,2$. Note that for the initialization phase, we omit such superscript since the phase is identical for both executions. Furthermore, for the variables computed by the Prover in the Response phase, the only difference comes from the use of different challenges $\gamma^{(i)}$ in the computation.

Under the assumption that the hash function $\mathbb{H}$ is collision resistant, for all $i \in \{0,\ldots,3\}$, we have $\mathfrak{f}_i^{(1)} = \mathfrak{d} = \mathfrak{f}_i^{(2)}$ and $\mathfrak{f}^{(1)} = \mathfrak{d} = \mathfrak{f}^{(2)}$.

Hence, for all $i \in \{0,\ldots,3\}$, we have $\mathfrak{g}^{z_i^{(1)} - z_i^{(2)}} \mathfrak{h}^{t_i^{(1)} - t_i^{(2)}} = \mathsf{c}_{y_i}^{\gamma^{(1)} - \gamma^{(2)}}$. Define $y_i' = \frac{z_i^{(1)} - z_i^{(2)}}{\gamma^{(1)} - \gamma^{(2)}}$ and $r_{y_i}' = \frac{t_i^{(1)} - t_i^{(2)}}{\gamma^{(1)} - \gamma^{(2)}}$, where division is performed over $\mathbb{Z}_p$, both values can be efficiently computed from the two accepting transcripts, and we obtain $\mathsf{c}_{y_i} = \mathfrak{g}^{y_i'} \mathfrak{h}^{r_{y_i}'}$.

Thus, for each $i \in \{1,2,3\}$, (y_i', r_{y_i}') is a valid opening of c_{y_i}. For $i = 0$, since $y_0 = w_1$ as set in Protocol 3, (y_0', r_{y_0}') is also a valid opening of c_{w_1}.

Now, since $\mathfrak{f}^{(1)} = \mathfrak{f}^{(2)}$, we have $\mathfrak{h}^{\tau^{(1)} - \tau^{(2)}} \mathfrak{g}^{(\gamma^{(1)} - \gamma^{(2)})} (\mathsf{c}_{w_2})^{4(z_0^{(1)} - z_0^{(2)})} = \prod_{i=1}^{3} (\mathsf{c}_{y_i})^{(z_i^{(1)} - z_i^{(2)})}$, which implies that we have $\mathfrak{g}^{\Phi_1} = \mathfrak{h}^{\Phi_2}$, where

$$\Phi_1 = (\gamma^{(1)} - \gamma^{(2)}) + 4w_2(z_0^{(1)} - z_0^{(2)}) - \sum_{i=1}^{3} y_i'(z_i^{(1)} - z_i^{(2)})$$

and

$$\Phi_2 = \sum_{i=1}^{3} r_{y_i}'(z_i^{(1)} - z_i^{(2)}) - (\tau^{(1)} - \tau^{(2)}) - 4r_{w_2}(z_0^{(1)} - z_0^{(2)}).$$

By the discrete log assumption, we have $\Phi_1 \equiv \Phi_2 \equiv 0 \pmod{p}$. First, this implies that we can calculate the following:

$$w_2' \equiv \frac{\sum_{i=1}^{3} y_i'(z_i^{(1)} - z_i^{(2)}) - (\gamma^{(1)} - \gamma^{(2)})}{4(z_0^{(1)} - z_0^{(2)})} \pmod{p}$$

and

$$r_{w_2}' \equiv \frac{\sum_{i=1}^{3} r_{y_i}'(z_i^{(1)} - z_i^{(2)}) - (\tau^{(1)} - \tau^{(2)})}{4(z_0^{(1)} - z_0^{(2)})} \pmod{p}.$$

This gives a valid opening of c_{w_2}. Secondly, from $\Phi_1 \equiv 0 \pmod{p}$, we also get $(\gamma^{(1)} - \gamma^{(2)})^2 + 4w_2'(z_0^{(1)} - z_0^{(2)})(\gamma^{(1)} - \gamma^{(2)}) \equiv \sum_{i=1}^{3} (z_i^{(1)} - z_i^{(2)})^2 \pmod{p}$. By our assumption on no wrap-around during the computation, the equality holds over the integers. Hence, dividing the equality by $(\gamma^{(1)} - \gamma^{(2)})^2$, we have

$$1 + 4w_1'w_2' = \sum_{i=1}^{3} (y_i')^2.$$

This shows that given two accepting transcripts with identical initialization phase, one can generate valid witness $(w_1', w_2', r_{w_1}', r_{w_2}')$ to $(\mathsf{c}_{w_1}, \mathsf{c}_{w_2})$ such that it belongs to $\mathcal{R}_{\mathsf{MultNNP}}$, proving that Protocol 3 satisfies the 2-special soundness.

Honest-verifier zero-knowledge (HVZK). The HVZK property follows from the standard simulation strategy for Σ-protocols. Given a challenge, the simulator samples the responses uniformly at random and computes the first message so that the verification equations hold. The resulting transcript is distributed identically to a real execution due to the linear structure of the protocol. This argument follows directly from the CKLR-style construction.

□

E. Full Specification of ZKRP by CKLR

In Protocol 4, we provide a full specification of ZKRP as proposed in [4, Section 5.3].

ZKRP requires public parameters $\mathsf{pp} = (\mathfrak{g}, \mathfrak{h}, q)$ for a group $\mathbb{G}$ of prime order q, where $\mathfrak{g}$ and $\mathfrak{h}$ are two generators of $\mathbb{G}$. We denote by λ, the security parameter, by $\mathbb{H}$ a collision-resistant hash function with 2λ-bit output, and by $S = 2^{2\lambda}$ and $C = L = 2^{\lambda}$ the parameters used by the protocol for the size of various randomness.

ZKRP gets as input a secret value committed with the Pedersen commitment scheme and a non-negative random mask of at most S.

The ZKRP protocol follows four steps. The prover performs some initial calculation through the initialization process. Then the verifier sends a challenge γ to the prover. Following this, the prover addresses the challenge and sends its response. Finally, the verifier verifies the prover's claim with its response and initial calculation. ZKRP and ZK-MultNNP perform the same four steps, with specific computation varying depending on the protocol.

F. Complexity Details

To facilitate the comparison between the complexity of Persephone and the baseline, we provide the complexity of the different components of an instantiation of a ZKRP or a ZK-MultNNP protocol in Table VI. The complexity of the prover and verifier is measured based on the building blocks in Table V.

Notation	Complexity for
$C_{\mathtt{Rand}}(\ell)$	a random generation of an ℓ-bit value
$C_{\mathtt{GExp}}(\ell)$	a group exponentiation by an ℓ-bit positive integer
$C_{\mathtt{GMult}}$	a group multiplication of two group elements
$C_{\mathtt{GInv}}$	a group inversion
$C_{\mathtt{3SP}}(\ell)$	a three-square decomposition of an ℓ-bit positive integer
$C_{\mathbb{H}}(n)$	a hash function evaluated on n concatenated group elements

TABLE V
SUMMARY TABLE OF THE COMPLEXITY NOTATIONS USED IN THIS WORK

The realizations of ZKRP and ZK-MultNNP in this section follow the realizations we previously discussed. More specifically, they consist of four phases: initialization, challenge, response, and verification as presented in Protocol 3 for ZK-MultNNP and Protocol 4 for ZKRP. We add a label "(P)" and "(V)" to signify whether a component is executed by the prover or the verifier. As previously argued, since the response phase only contains integer computations, its complexity is negligible relative to the complexity of other building blocks, including group operations and the generation of random numbers.

ZKRP	Initialization (P)	$3C_{\mathtt{Rand}}(2\lambda) + 4C_{\mathtt{Rand}}(4\lambda) + 4C_{\mathtt{Rand}}(b+2\lambda) + 3C_{\mathtt{Rand}}(b+4\lambda+2) + C_{\mathtt{3SP}}(2b+2) + 3C_{\mathtt{GInv}} + 11C_{\mathtt{GMult}} + 3C_{\mathtt{GExp}}(2\lambda) + 4C_{\mathtt{GExp}}(4\lambda) + 3C_{\mathtt{GExp}}(b) + 7C_{\mathtt{GExp}}(b+2\lambda) + C_{\mathtt{GExp}}(b+2\lambda+2) + C_{\mathtt{GExp}}(b+4\lambda+2) + C_{\mathbb{H}}(5)$.
	Challenge (V)	$C_{\mathtt{Rand}}(\lambda)$
	Response (P)	Negligible
	Verification (V)	$5C_{\mathtt{GInv}} + 14C_{\mathtt{GMult}} + 5C_{\mathtt{GExp}}(\lambda) + 4C_{\mathtt{GExp}}(4\lambda) + C_{\mathtt{GExp}}(b) + 7C_{\mathtt{GExp}}(b+2\lambda) + C_{\mathtt{GExp}}(b+2\lambda+2) + C_{\mathtt{GExp}}(b+4\lambda+2) + C_{\mathbb{H}}(5)$
	Verifier communication	λ
	Prover communication	$3\mathbb{G} + 5b + 30\lambda + 2$
ZK-MultNNP	Initialization (P)	$3C_{\mathtt{Rand}}(2\lambda) + 4C_{\mathtt{Rand}}(4\lambda) + 4C_{\mathtt{Rand}}(b+2\lambda) + C_{\mathtt{Rand}}(b+4\lambda+2) + C_{\mathtt{3SP}}(2b+2) + 3C_{\mathtt{GInv}} + 11C_{\mathtt{GMult}} + 3C_{\mathtt{GExp}}(2\lambda) + 4C_{\mathtt{GExp}}(4\lambda) + 3C_{\mathtt{GExp}}(b) + 7C_{\mathtt{GExp}}(b+2\lambda) + C_{\mathtt{GExp}}(b+2\lambda+2) + C_{\mathtt{GExp}}(b+4\lambda+2) + C_{\mathbb{H}}(5)$.
	Challenge (V)	$C_{\mathtt{Rand}}(\lambda)$
	Response (P)	Negligible
	Verification (V)	$4C_{\mathtt{GInv}} + 13C_{\mathtt{GMult}} + 5C_{\mathtt{GExp}}(\lambda) + 4C_{\mathtt{GExp}}(4\lambda) + 7C_{\mathtt{GExp}}(b+2\lambda) + C_{\mathtt{GExp}}(b+2\lambda+2) + C_{\mathtt{GExp}}(b+4\lambda+2) + C_{\mathbb{H}}(5)$.
	Verifier communication	λ
	Prover communication	$3\mathbb{G} + 5b + 30\lambda + 2$

TABLE VI
WORST-CASE COMPLEXITY FOR ZKRP AND ZK-MULTNNP

With the complexity of ZKRP and ZK-MultNNP, we perform the complexity analysis of Persephone and the baseline, and report it in Table VII and Table VIII. At initialization, the baseline does nothing. On the other hand, Persephone requires 2 calls of ZK-MultNNP to show that $0 \leq l_0 \leq u_0 \leq B$. Hence, we provide the complexity summary at initialization of Persephone in Table VII. In terms of verifier storage, we assume that only verified commitments are stored, i.e. $(1\|\mathsf{c}_x)$, which adds two more bits to the verifier communication. At other time periods, we provide directly the complexity summary table VIII, which can be derived directly from Protocol 1.

Protocol 4 Realization of ZKRP

Parameters: Security parameter λ, Upper bound parameters: random mask bound $S = 2^{2\lambda}$, challenge bound $C = 2^{\lambda}$, and additional security padding parameter $L = 2^{\lambda}$.

Inputs: A public commitment c_x of an input x with associated random masks r_x. The input and private random mask are held by the prover, while the public commitment is known by both prover and verifier.

Output: If $0 \leq x \leq B$, the verifier accepts with probability 1.

Initialization

Prover

1: Set $y_0 = B - x$ and $r_{y_0} = -r_x$
2: Compute y_1, y_2, and y_3 with bit length $(2 + \lceil \log(x(B - x)) \rceil)/2$ s.t. $(y_1, y_2, y_3) = \mathcal{F}^{(3\mathsf{SP})}(4x(B-x)+1)$
3: For $i \in \{1,2,3\}$, randomly sample $r_{y_i} \in [0, S]$
4: For $i \in \{1,2,3\}$, compute $\mathsf{c}_{y_i} = \mathfrak{g}^{y_i}\mathfrak{h}^{r_{y_i}}$
5: For $i \in \{0,1,2,3\}$, randomly sample $m_i \in [0, BCL]$
6: For $i \in \{0,1,2,3\}$, randomly sample $s_i \in [0, SCL]$
7: For $i \in \{0,1,2,3\}$, compute $\mathfrak{d}_i = \mathfrak{g}^{m_i}\mathfrak{h}^{s_i}$
8: Randomly sample $\sigma \in [0, 4SBCL]$
9: Compute $\mathfrak{d} = \mathfrak{h}^{\sigma}(\mathsf{c}_x)^{4m_0}\mathsf{c}_{y_1}^{-m_1}\mathsf{c}_{y_2}^{-m_2}\mathsf{c}_{y_3}^{-m_3}$
10: Compute $\Delta = \mathbb{H}(\mathfrak{d}_0 \| \mathfrak{d}_1 \| \mathfrak{d}_2 \| \mathfrak{d}_3 \| \mathfrak{d})$

$\mathsf{c}_{y_1}, \mathsf{c}_{y_2}, \mathsf{c}_{y_3}, \Delta$ ⟶

Challenge

Verifier

1: Randomly sample $\gamma \in [0, C]$

⟵ γ

Response

Prover

1: For $i \in \{0,1,2,3\}$, compute $z_i = m_i + \gamma y_i$ and $t_i = s_i + \gamma r_{y_i}$
2: Compute $\tau = \sigma + \gamma\left(\sum_{i=1}^{3} y_i r_{y_i} - 4y_0 r_{y_0}\right)$

$z_0, z_1, z_2, z_3, t_0, t_1, t_2, t_3, \tau$ ⟶

Verification

Verifier

1: **for** $i \in \{1,2,3\}$ **do**
2: **if** $z_i \notin [0, BC(L+1)]$ **then**
3: Return 0 ▷ the verification failed, z_i is not sufficiently small.
4: Compute $\mathsf{c}_{y_0} = \mathsf{c}_x^{-1}\mathfrak{g}^{B}$;
5: **for** $i \in \{0,1,2,3\}$ **do**
6: Compute $\mathfrak{f}_i = \mathfrak{g}^{z_i}\mathfrak{h}^{t_i}\mathsf{c}_{y_i}^{-\gamma}$
7: Compute $\mathfrak{f} = \mathfrak{h}^{\tau}\mathfrak{g}^{\gamma}(\mathsf{c}_x)^{4z_0}\prod_{i=1}^{3}\mathsf{c}_{y_i}^{-z_i}$
8: **if** $\Delta = \mathbb{H}(\mathfrak{f}_0 \| \mathfrak{f}_1 \| \mathfrak{f}_2 \| \mathfrak{f}_3 \| \mathfrak{f})$ **then**
9: Return 1 ▷ the verification is successful, c_x is a valid commitment of x, $\mathfrak{f}_i = \mathfrak{d}_i$, and $4x(B-x)+1 = y_1^2 + y_2^2 + y_3^2$, $\mathfrak{f} = \mathfrak{d}$.
10: **else**
11: Return 0 ▷ the verification failed

Verifier computation complexity	$2C_{\texttt{Rand}}(\lambda) + 9C_{\texttt{GInv}} + 28C_{\texttt{GMult}} + 10C_{\texttt{GExp}}(\lambda) + 8C_{\texttt{GExp}}(4\lambda) + C_{\texttt{GExp}}(b) + 14C_{\texttt{GExp}}(b + 2\lambda) + 2C_{\texttt{GExp}}(b + 2\lambda + 2) + 2C_{\texttt{GExp}}(b + 4\lambda + 2) + 2C_{\mathbb{H}}(5)$
Prover computation complexity	$8C_{\texttt{Rand}}(2\lambda) + 8C_{\texttt{Rand}}(4\lambda) + 8C_{\texttt{Rand}}(b + 2\lambda) + 2C_{\texttt{Rand}}(b + 4\lambda + 2) + 2C_{\texttt{3SP}}(2b + 2) + 6C_{\texttt{GInv}} + 24C_{\texttt{GMult}} + 8C_{\texttt{GExp}}(2\lambda) + 8C_{\texttt{GExp}}(4\lambda) + 8C_{\texttt{GExp}}(b) + 14C_{\texttt{GExp}}(b + 2\lambda) + 2C_{\texttt{GExp}}(b + 2\lambda + 2) + 2C_{\texttt{GExp}}(b + 4\lambda + 2) + 2C_{\mathbb{H}}(5)$
Verifier communication complexity	$2\mathbb{G} + 2\lambda + 2$
Prover communication complexity	$8\mathbb{G} + 10b + 60\lambda + 4$

TABLE VII
WORST-CASE COMPLEXITY OF PERSEPHONE AT INITIALIZATION

Verifier computation complexity	Baseline	$4C_{\texttt{Rand}}(\lambda) + 20C_{\texttt{GInv}} + 58C_{\texttt{GMult}} + 20C_{\texttt{GExp}}(\lambda) + 16C_{\texttt{GExp}}(4\lambda) + 4C_{\texttt{GExp}}(b) + 28C_{\texttt{GExp}}(b + 2\lambda) + 4C_{\texttt{GExp}}(b + 2\lambda + 2) + 4C_{\texttt{GExp}}(b + 4\lambda + 2) + 4C_{\mathbb{H}}(5)$
	PERSEPHONE	$3C_{\texttt{Rand}}(\lambda) + 8C_{\texttt{GInv}} + 25C_{\texttt{GMult}} + 7C_{\texttt{GExp}}(\lambda) + 2C_{\texttt{GExp}}(3\lambda + 1) + 4C_{\texttt{GExp}}(4\lambda) + 8C_{\texttt{GExp}}(b) + 7C_{\texttt{GExp}}(b + 2\lambda) + C_{\texttt{GExp}}(b + 2\lambda + 2) + C_{\texttt{GExp}}(b + 4\lambda + 2) + C_{\mathbb{H}}(5)$
Prover computation complexity	Baseline	$15C_{\texttt{Rand}}(2\lambda) + 16C_{\texttt{Rand}}(4\lambda) + 16C_{\texttt{Rand}}(b + 2\lambda) + 12C_{\texttt{Rand}}(b + 4\lambda + 2) + 4C_{\texttt{3SP}}(2b + 2) + 12C_{\texttt{GInv}} + 47C_{\texttt{GMult}} + 15C_{\texttt{GExp}}(2\lambda) + 16C_{\texttt{GExp}}(4\lambda) + 15C_{\texttt{GExp}}(b) + 28C_{\texttt{GExp}}(b + 2\lambda) + 4C_{\texttt{GExp}}(b + 2\lambda + 2) + 4C_{\texttt{GExp}}(b + 4\lambda + 2) + 4C_{\mathbb{H}}(5)$
	PERSEPHONE	$6C_{\texttt{Rand}}(2\lambda) + 2C_{\texttt{Rand}}(3\lambda + 1) + 4C_{\texttt{Rand}}(4\lambda) + 4C_{\texttt{Rand}}(b + 2\lambda) + C_{\texttt{Rand}}(b + 4\lambda + 2) + C_{\texttt{3SP}}(2b + 2) + 3C_{\texttt{GInv}} + 14C_{\texttt{GMult}} + 6C_{\texttt{GExp}}(2\lambda) + 2C_{\texttt{GExp}}(3\lambda + 1) + 4C_{\texttt{GExp}}(4\lambda) + 6C_{\texttt{GExp}}(b) + 7C_{\texttt{GExp}}(b + 2\lambda) + C_{\texttt{GExp}}(b + 2\lambda + 2) + C_{\texttt{GExp}}(b + 4\lambda + 2) + C_{\mathbb{H}}(5)$
Verifier communication complexity	Baseline	4λ
	PERSEPHONE	$2\mathbb{G} + 3\lambda + 2$
Prover communication complexity	Baseline	$15\mathbb{G} + 20b + 120\lambda + 8$
	PERSEPHONE	$8\mathbb{G} + 5b + 36\lambda + 4$

TABLE VIII
WORST-CASE COMPLEXITY OF PERSEPHONE AND THE BASELINE AT TIMESTAMP $t \geq 1$